On the Design of End-user Service Composition Applications

Ridge, Andrew

Award date:
2015

Awarding institution:
University of Bath

On the Design of End-user Service Composition Applications

submitted by

Andrew David Ridge

for the degree of Doctor of Philosophy

of the

University of Bath

Department of Computer Science

August 2014

ABSTRACT

End-user Service Composition (EUSC) is a relatively new field that aims to enable non-developers to create bespoke applications and services by coordinating multiple component services created by a range of different developers. One of the main challenges of EUSC is that it is an instance of an ill-structured problem: a problem with multiple solutions, multiple paths to those solutions, and no consensus as to which solution might be best [Jonassen, 1997]. We suggest that design spaces are an effective method for navigating ill-structured problems such as EUSC. Design spaces are multi-dimensional spaces where dimensions represent prospective design decisions, and points on those dimensions represent potential solutions to those decisions. The work in this thesis aims to explore how design spaces can be used in design generation in software engineering, and in particular in the domain of EUSC.

Building on the literature we identified three research goals: (i) to derive and evaluate a set of requirements for an EUSC application, (ii) to create and evaluate a concrete design space for EUSC applications, and (iii) to implement and evaluate a software tool to allow designers to create and interact with concrete design spaces.

Whilst solving these goals, we contributed two large bodies of knowledge to the EUSC domain: a set of 139 requirements for an EUSC application, and a concrete design space for EUSC applications containing over 600 design elements. To derive the requirements, we created a bespoke method aimed at gathering requirements from end-users, based on established methods. To create the EUSC design space, we first clarified and extended the vocabulary of the domain, before specifying our own design space creation method where none existed previously. To allow designers to interact with design spaces, we developed a design space tool that supported the creation of design spaces, profiling applications in the domain, and the generation of new designs. Finally, we explored the use of the design space in design generation, where novice designers were tasked with generating a design for an EUSC application. Our findings showed that increasing design space support resulted in designs that were more balanced, more concrete, more complex, and most intriguingly, less novel for the domain. We provide recommendations as to how designers can balance this trade off, and use design spaces to create EUSC applications as well as other instances of ill-structured problems.

ACKNOWLEDGEMENTS

I would like to take this opportunity to thank my supervisor Prof. Eamonn O'Neill for his support throughout the past few years. I would also like to thank Dr Ryan Kelly and Dr Tom Lovett for their feedback, particularly in the latter stages of this work. For their support throughout my PhD, thanks also go to my colleagues (soon to be Dr) David Wilson, (soon to be Dr) Michael Wright, (soon to be Dr) Andrew Chinery, Jim Grimmett, Tom Fletcher and Ieuan Evans.

I would also like to thank everyone in the department, the employees at the coffee shop, and the members of the SU wine society for keeping me sane through the past four years.

Finally, I would like to thank Benita and my family for their love, support and patience throughout my PhD, and perhaps most importantly, for not asking how it was going too often.

TABLE OF CONTENTS

LIST OF FIGURES

LIST OF TABLES

GLOSSARY

Definition 1: Service Composition (SC).
The process of coordinating together a collection of component services to create a composite service.

Definition 2: End-user Service Composition (EUSC).
A specialisation of SC where the user who executes the composite is (or has the potential to be) the same user who composed it. They are also expected to be a consumer rather than a business user.

Definition 3: Component Service.
A service that is an input to the composition process, i.e. one of the services that is coordinated together.

Definition 4: Composite Service.
The output of the composition process; a coordinated collection of component services.

Definition 5: EUSC Application.
An application created by a developer that allows an end-user to perform composition and interact with the composite afterwards.

Definition 6: Mashup.
A Web-based service that integrates a number of separate data sources, interfaces, or processes (analogous to a composite service).

Definition 7: Mashup Development Environment (MDE).
An EUSC application specific to the mashup domain.

Definition 8: Design Space (DS).
A multi-dimensional space where dimensions represent design decisions, and points on those dimensions represent potential solutions to the corresponding design decision.

Definition 9: Design decision.
A decision that may need to be considered in the design of an artefact.

Definition 10: Design solution.
An alternative that, if chosen, is a resolution to a design decision.

Definition 11: Design element.

A design decision, or a design solution.

CHAPTER 1
INTRODUCTION

1.1 Thesis Overview

End-user Service Composition (EUSC) is a process by which end-users can create small, bespoke services or applications by combining services created by developers. There are many ways of achieving EUSC, and it is not clear which ways are best, meaning that EUSC falls into the category of ill-structured problems [Jonassen, 1997]. In this thesis, we aim to explore how design spaces can be used to motivate the design of EUSC applications and applications in other ill-specified domains.

This thesis investigates how design spaces can be used to inform the design of End-user Service Composition (EUSC) applications. Service Composition (SC) is a process where users of indeterminate technical ability can coordinate together a collection of component services to create a composite service – typically with much more useful functionality than any of the separate components. EUSC is a specialisation of this process where the user performing the composition is the same user as the executor of the composite service, and is typically thought to be a consumer rather than a business user.

To motivate the problem that EUSC aims to solve, consider a motivating scenario:

"Ben has a London Underground tube status application for his smartphone that allows him to check the status of any tube line. He identifies that if he checks the status of the Bakerloo line at the time he normally wakes up, the delays may cause him to be late for work. Instead, he would like to be woken early when there are delays on the Bakerloo line, and at his normal time when the lines are not delayed."

"He composes a tube status service with an alarm service and a timer service. The timer service activates at 6am, signals the tube status service to check the status of the lines, and signal the alarm service to set off an alarm if there are delays on the Bakerloo line."

EUSC applications are becoming increasingly popular and a number of commercially available EUSC applications have been released recently, particularly with the rising popularity of Android, Web APIs, and more recently, home automation and the Internet of Things.

Currently there is little consensus as to how EUSC applications should operate, what they should do, how the user should interact with them, how they are implemented, as well as many other issues. This is because EUSC is an instance of an ill-structured problem: a problem with multiple valid solutions, with multiple ways of reaching these solutions, and no consensus as to which solution might be best [Jonassen, 1997].

We feel that the design of EUSC applications and other instances of ill-structured problems would benefit from the structured approach that is provided by design spaces. Design spaces are multi-dimensional spaces that can be used to describe the set of design choices that have been made, or could be made, in the design of an artefact. Each dimension in the space represents a prospective design decision, and values on those dimensions reflect potential solutions to that design decision. Hence, one can consider a design artefact as being a single point in this multi-dimensional space.

Several authors have independently created design spaces for EUSC applications (or closely related Mashup Development Tools): [Aghaee et al., 2012, Grammel and Storey, 2010, Minhas et al., 2012, Na et al., 2010, Mehandjiev and De Angeli, 2012, Albinola et al., 2009, Fischer et al., 2009, Pietschmann et al., 2010, Brønsted et al., 2010, Tuchinda et al., 2011]. The EUSC design spaces were created to meet different aims, but on the whole, the design spaces created in these works are limited in both size and scope. Most of the work in which the design spaces appear often does not provide details that are useful to other researchers in the area, in particular the method by which the design space was created. None of these prior design spaces investigate how design spaces can be used to influence design in the software engineering process.

The aim of this thesis is to explore how design spaces can be used to inform the process of designing EUSC applications, and how best to support the use of design spaces in this process. In the remainder of this section we provide further discussion of EUSC and design spaces to help scope the research goals and objectives that are presented later in this chapter.

1.1.1 End-user Service Composition

End-user Service Composition (EUSC) is a specialisation of Service Composition (SC). SC is a promising area that allows non-developers to coordinate component software services together in order to create a more useful, powerful, and usually more complex application or service. There are four high-level activities that need to be supported to enable SC, listed below and illustrated in Figure 1-1:

(a) Designers of SC applications create an SC application.
(b) Service developers creating atomic services to be used within an SC application.
(c) The user of the SC application (created in step *(a)*) combines multiple atomic services (created in step *(b)*) in order to create a new, composite service.

(d) The user of the SC application uses the composite (created in step *(c)*) to perform some desired action.

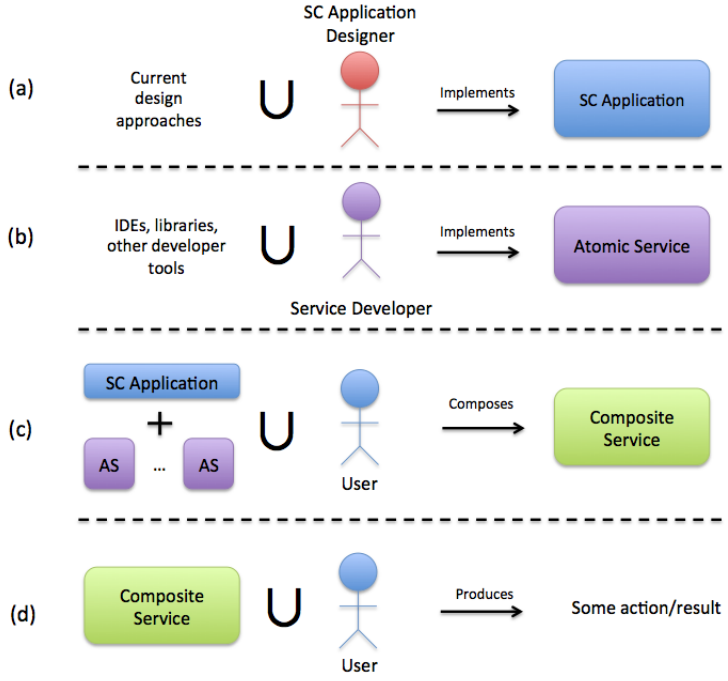


Figure 1-1: A high-level overview of Service Composition (IDE = integrated development environment, AS = atomic service).

Figure 1-1 shows the four main stages in SC, as well as indicating the user who takes part in that stage, the resources that they use in this stage, and the result of the stage. The research in this thesis focuses on EUSC, which is SC where the user who creates the composite in stage *(c)* is the same as the user who executes it in stage *(d)*.

EUSC allows normal consumers with low technical skill to create small, customised applications (composite services) by coordinating together a collection of components that are created by developers. These end-users interact with an EUSC application to discover and coordinate the components in order to create the composite that they require [Obrenović and Gašević, 2008].

Developing EUSC applications is a challenge because SC is an inherently technical process, and conveying this process to end-users in a way that allows them to understand what they can achieve through it is important. These challenges impact the design choices that the designer of an EUSC application can make, such as restricting the types of component that can be coordinated, or the number of components that can be connected together at once.

Mashups, and particularly end-user development (EUD) of mashups, is another sub-field of SC that is closely related to EUSC. In this thesis, we consider mashups as being a specific instance of EUSC applications that focus on the Web [Aghaee and Pautasso, 2014]. We

consider EUSC in the general case, and provide examples of EUSC applications across the Web, as well as those available on the desktop, and mobile applications. Mashup development tools were commercially popular circa 2007, when a number were released by large technology companies such as Yahoo!, Google, Microsoft, Intel, and IBM. All except Yahoo! Pipes[1] have since been discontinued. Conversely, new EUSC applications have been released throughout the time of writing this thesis (particularly on Android), with one such example gaining popularity across technology blogs: If This Then That (IFTTT)[2], and its associated mobile applications available on iOS[3] and Android[4].

In the literature review (Chapter 2), we provide more detail as to how EUSC can be achieved, who can be involved in the process, as well as providing more discussion into the difference between EUSC, Mashups, and End-user Development.

1.1.2 Design Spaces

Design spaces were first suggested by [Lane, 1990] as a method for recording and classifying design choices made for an artefact. A number of other authors have provided other definitions for 'design space' [MacLean and McKerlie, 1995, Baum et al., 2000, Wood and Agogino, 2005, Westerlund, 2005], all of which are variations upon the theme of a design space being a multi-dimensional space of design decisions and their corresponding solutions.

The term design space can also be used in a metaphorical sense, where an author refers to 'the design space for X' [Szafir and Mutlu, 2012, Kriplean et al., 2012, Pierce, 2012, Parker et al., 2012]. Only Gooch [Gooch, 2013] has sought to provide more insightful classification of what design spaces are and how they can be used: conceptual, evaluative, and generative. Whilst these classifications are a useful starting point for our work, we believe that further clarification is required to define what design spaces are, and where they should be used in the software engineering process. Software support is cited as being an important aspect of design spaces due to their size and complexity [Baum et al., 2000], but it is not clear how such software can be used to support designers, and exactly what tasks are required of them.

We believe that design spaces are helpful in solving ill-structured problems because design spaces are inherently structured and can help designers and software engineers approach these problems in a systematic way. Ill-structured problems are types of problems with multiple solutions and multiple paths to those solutions where no consensus exists as to the concepts that solve the problem best [Jonassen, 1997]. EUSC clearly meets this definition, and we believe that using design spaces to support the design process for EUSC applications will be beneficial in solving such problems. To achieve this, we also need to better understand

[1] http://pipes.yahoo.com
[2] http://ifttt.com
[3] https://itunes.apple.com/gb/app/ifttt/id660944635?mt=8
[4] https://play.google.com/store/apps/details?id=com.ifttt.ifttt

what design spaces are, how they can be used, and the effect that their use can have on the design process.

1.2 Problem Statement and Research Goals

The work listed above does not motivate any concrete research questions, and as such the work in this thesis is based on three high-level research goals.

Problem statement: EUSC applications are a class of application that can be used to achieve a potentially endless set of user goals, and composition can be performed in a myriad of different ways – both technically, and in its representation to the user. We aim to support the design of EUSC applications through the use of design spaces and evaluate how design spaces can aid design in the software engineering process, what support is required to use them in the software engineering process, and to provide insight into the design of EUSC applications.

In order to be able to consider the design of such applications, we need a set of requirements to which an EUSC application must adhere. Thus, our first research goal is:

> **RG1:** Derive and evaluate a set of requirements for an EUSC application.

Once we have identified the requirements that underpin an EUSC application, we will be in a position to consider the design choices to be made for EUSC applications. This motivates our second research goal:

> **RG2:** Create and evaluate a Design Space for EUSC applications.

Design spaces are large, monolithic entities that can be difficult to interact with. It has been suggested that tool support is necessary in order to facilitate interaction with design spaces. This leads to our final research goal:

> **RG3:** Create and evaluate a Design Space Tool to facilitate the design and creation of design spaces and domain applications.

Once we have achieved these aims, we will have both extended the knowledge in the EUSC domain, demonstrated how to support the use of design spaces, and provided insight as to how design spaces can affect design in the software engineering process.

1.3 Research Methods

We use a mixed methods approach in this thesis, utilising a number of data collection and analysis methodologies. We present two empirical studies: a requirements gathering study (Chapter 3), and a study to evaluate the use of the design space tool in design generation

(Chapter 6), both of which resulted in large sets of qualitative data, as well as limited amounts of quantitative data and quantitative metadata.

The requirements gathering study uses a method that is based on an established requirements gathering technique in the data gathering stage: the Scenario-based Requirements Analysis Method (SCRAM) [Sutcliffe, 2003, Sutcliffe and Ryan, 1998]. The output of the sessions were videos, which were subsequently transcribed. In the data analysis stage, we use a content analysis technique based on themes already identified in the domain: Directed Content Analysis (DCA) [Hsieh and Shannon, 2005]. The output of this content analysis was subsequently transformed into a set of requirements.

The design space exploration study required users to enter design choices directly into a Web-based application. This application was linked to a MySQL database where participants design choices were recorded. This yielded a large amount of qualitative data that were transformed and analysed to assess how the design space affected the process of generating the design. A detailed description of the analysis is provided in Section 6.3 (page 167).

Our design space creation method is described in Chapter 4 and is a bespoke method based on stages identified in prior work on design spaces. The data used in this study were derived from prior literature in the domain, domain applications, and the results of our two studies.

1.3.1 Research Ethics

Both of the empirical studies in this thesis adhered to the 13-point ethics check list created by the Computer Science Department of the University of Bath. The check list was filled in prior to each study being carried out, and no issues were found in either case.

1.4 Thesis Organisation and Contributions

Chapter 1: Introduction The introduction has introduced the problem areas of this thesis: EUSC and design spaces, as well as motivating our investigation into the exploration of these two problem areas.

Chapter 2: Service Composition and Design Background In the literature review, we provide relevant background in the two main research areas that are covered in this thesis: Service Composition (SC) and design spaces. Our discussion of SC first introduces Services, before discussing a number of topics related to SC, as well as related areas such as Mashups, EUSC and End-user Development (EUD). The second part of the literature review frames our work on Design Spaces, by presenting a discussion of what they are and how they can be used. We then clarify similar aspects of the software engineering process, before discussing design and creativity.

Chapter 3: Establishing Requirements for EUSC Applications In Chapter 3, we derive a set of requirements for an EUSC application. Prior work with the explicit aim of gathering EUSC requirements has yielded two small sets of requirements that are specific to sub-topics within EUSC. The chapter describes a study that was carried out using a bespoke requirements gathering method based on an adapted form of the Scenario-based Requirements Analysis Method (SCRAM) and Directed Content Analysis (DCA).

Our method extends existing work by adapting SCRAM such that it can be carried out with end-users rather than a business customer. We also extend the method to incorporate the elicitation of requirements into the method. Finally, we analyse the requirements to ensure that they were correct, consistent and complete.

The contributions of this work are two-fold:

- **Methodological:** An end-to-end, robust and repeatable requirements gathering method for gathering requirements from end-users.
- **Practical:** A set of requirements for EUSC applications.

Chapter 4: A Design Space for EUSC Applications This chapter has two main aims. In the first section of this chapter, we discuss various definitions of 'design space' that are presented in the literature, before providing a clarification of some of the definitions of types of design space to provide a structure for researchers and designers to clarify exactly what they mean when they refer to a 'design space'. We also specify the tasks design spaces can support, and where they should be used in the software engineering process.

Our next aim is to create an explicit design space for EUSC applications, however there is little guidance in prior work as to how a design space can be created in a particular domain. Thus, we document the process of creating our design space for EUSC applications and hence provide a repeatable and generalisable method for creating explicit design spaces in a given domain. We provide an analysis of this method throughout the discussion, providing insight as to the features that a domain might need for our method to be applicable.

This chapter has two contributions:

- **Methodological:** A generalisable, repeatable method for creating design spaces in any domain.
- **Practical:** A design space for EUSC Applications.

Chapter 5: Design Space Tool Tool support is cited as being very important for using explicit design spaces given their complexity and size [Baum et al., 2000]. We discuss the creation and analytical evaluation of a design space tool based on three use cases for working with generic design spaces: (1) design space creation, (2) application profiling (classifying chosen design choices for) existing applications in the domain using the design space, and (3) using the design space tool to generate new designs in the domain. Our design space

tool is grounded in the design space for EUSC applications. We also present the results of an analytical evaluation of the design space tool using Hierarchical Task Analysis (HTA) and Cognitive Walkthrough (CW). Our evaluation is based on the design space creation and design space profiling tasks as they are well-defined and rigidly structured.

The contribution of this chapter is:

- **Practical:** The creation and evaluation of a design space tool for Design Spaces that is demonstrated using the explicit design space created in the previous chapter.

Chapter 6: Design Space Study We present an empirical study exploring the effects of providing designers with an explicit design space when they generate a design for an EUSC application. The aim of the chapter is to evaluate the effect of design space support on the designs that are created by novice designers who are unfamiliar with the domain. The study varied the level of design space support that participants received, as well as the representation of the design space that they were able to use, and asked them to record the design choices that they would make if asked to create an EUSC application. Our findings indicated that the representation of the design space had little effect on the designs that participants created, whereas the level of design space support had a number of interesting effects such as the number of novel decisions, the generality of the design, and the design's balance/structure.

Finally, we provided a set of recommendations for how these findings can be used to support the use of design spaces in the software engineering process. The contribution of this chapter is:

- **Theoretical:** The results of our design space study indicate that increased design space support yields designs that are more specific, more concrete, more complex, more balanced, but less novel.

Chapter 7: Conclusion The conclusion discusses each of the main findings of this thesis, as well as discussing the theoretical, methodological and concrete contributions of the thesis, before discussing limitations and future work.

CHAPTER 2
BACKGROUND & RELATED WORK

2.1 Chapter Introduction

This chapter discusses the two domains in which this thesis operates: End-user Service Composition (EUSC) and design spaces. Our discussion is split into three sections: first we discuss Service Composition (SC) in enough detail for the reader to be able to understand the terminology and implications throughout the rest of the thesis. Secondly, we discuss the current state of knowledge of design spaces, as well as relating design spaces to software engineering and design theory, so that the extensions we provide to this domain are clear. Finally, we discuss the intersection of these two areas by presenting a number of design spaces that have been created for applications in the EUSC domain.

2.1.1 Chapter Organisation

The first section of the literature review provides relevant background for EUSC by covering the following topics related to Service Composition (SC):

1. Service Composition
   (a) Entities
   (b) Process
   (c) Stakeholders
2. End-user Service Composition
   (a) Mashups
   (b) End-user Development
   (c) EUSC applications
   (d) Requirements for EUSC applications

These topics provide enough background for the reader to better understand what SC and EUSC are, as well as the discussing the stakeholders in these processes. We discuss other topics that are related to EUSC that have implications on our work, such as mashups and End-user development. We then discuss examples of EUSC applications that currently exist

in the domain – these are presented to participants in each of our studies – before moving on to discuss requirements that have been derived for EUSC applications previously.

We do not use the terms SC and EUSC interchangeably. The distinction between the two is important because there are certain assumptions made for EUSC that do not necessarily hold for SC, in particular the expected knowledge or skills of the user performing composition. In EUSC, it is assumed that the composer is not a developer, whereas this assumption is not made for SC. This is primarily a matter of specificity, but it becomes important in some sections where we describe some aspects of background work that relate to both SC and EUSC in different ways.

The second section of the literature review motivates our work on design spaces, and describing where design spaces fit within the field of design. This discussion is separated into the following topics:

1. Design spaces
   (a) Contents of design spaces
   (b) Design space usage
   (c) Design space creation
   (d) Design space evaluation
   (e) Limitations of design spaces
   (f) Design spaces in software engineering
2. Design
   (a) Decision-based Design
   (b) Design rationale
   (c) Creativity

These design space topics help us to clarify different aspects of design spaces that we can use later. As we will discuss, we do not feel that the definitions of design spaces provided in prior work are sufficient to describe how they can be used. We also provide some background that places design spaces within software engineering, the field of design, and other related areas such as design rationale and creativity as this discussion has implications for the work in this thesis.

Finally, we present and discuss the design spaces that have been created for EUSC applications that formed part of the motivation for our work.

2.2 Service Composition

Service Composition (SC) has a number of definitions, all of which are variations on a theme – SC is the process of creating composite services from a series of component services: *"[Dynamic] service composition is the process of creating new services [at runtime] from a set of service components"* [Tosic et al., 2000].

Other definitions of SC cover aspects of composition such as how it is achieved (e.g. via a scripting language [Laga and Bertin, 2012], or a graphical language [Casati, 2011]), the logical operations that are allowed (e.g. loops, forks, joins, etc. [Laga and Bertin, 2012]), the time that the composition is performed (i.e. design-time [static composition] or run-time [dynamic composition] [Ibrahim et al., 2010]), or the technologies that power the composition system (e.g. Web Services Business Process Execution Language [WS-BPEL] or the Web Services Choreography Description Language [WS-CDL] [Ro et al., 2008]). Rather than being integral to the definition of SC, we believe that each of these facets are properties or functions that *can* exist in SC.

We can distinguish between three different types of SC, based on the types of service that are being composed [Daniel et al., 2007, Picozzi, 2014]:

1. **Data integration**: Using composition to integrate a number of data sources, usually to manipulate the data.
2. **Application/process integration**: Using composition to coordinate a number of different applications or processes.
3. **Presentation/interface integration**: Using composition to connect together a number of user interface components.

SC can also be automated to various degrees, which is measured by the amount of control that the user has over the process [Zhang et al., 2003]. SC automation is split into three levels:

- **Automatic**: No user involvement at all [Aghaee et al., 2012].
- **Manual**: Full user involvement with no assistance from the system [Kapitsaki et al., 2007].
- **Semi-automatic**: Some user involvement with some assistance from the system, to lower the user burden [Aghaee et al., 2012].

Our work does not impose any constraints on the type of composition that is being performed since many of the considerations that apply to process composition would also apply to data or interface composition. However, we assume that the composition process is not fully automated since our focus is on EUSC, and automated composition does not require that a person coordinates the components with one another.

There are two types of 'architecture' that SC can follow: an orchestration or a choreography. An orchestrated composition approach is on where a single party (an orchestrator) is in control of the composition process, where each component executes and returns its result to the orchestrator [Peltz, 2003]. A choreography is a more collaborative approach, where components communicate directly with one another through self-described interfaces [Peltz, 2003, W3C, 2004b]. The intuition behind a choreographed composition approach is *"Dancers dance following a global scenario without a single point of control."*[5]. The work in

[5] `http://en.wikipedia.org/wiki/service_choreography`

this thesis does not distinguish between either of these approaches.

In the rest of this section, our discussion is framed around the following aspects of SC:

- **SC Entities**: The entities that the user interacts with in composition, i.e., the types of service that can be involved in the process.
- **The SC life cycle**: The steps that are involved in the SC process.
- **SC Stakeholders**: The actors that take part in the SC process, and the roles that they can fulfil.
- **End-user Service Composition**: A discussion of the differences between SC and EUSC, including motivations for it and problems with it.
  - **Mashups**: A closely related area to EUSC, based on the Web.
  - **End-user Development (EUD)**: The activity of getting end-users to create applications (EUSC can be thought of as a type of EUD).
  - **EUSC applications**: A discussion of some current examples of EUSC applications that are referred to throughout the thesis.
  - **Requirements for EUSC applications**: Examples of prior approaches to gathering requirements for EUSC applications.

2.3 Entities in Service Composition

The entities in SC are the services that the user of an SC application interacts with (i.e. the services in the composition process). Our main focus in this section is to describe what we mean when we use the term 'service'.

2.3.1 Services

The definition of service is very broad, and to better structure our discussion of what we mean by 'service', we discuss them in the context of generic services, before moving to a more specialised discussion of software services that could be used within an SC application.

Generic Services

The most basic definition of a *service* is that it is an intangible version of an economic good [Akehurst and Gadrey, 1988, Mehandjiev et al., 2010b]. Services can be found in great many domains, and in some domains services may blur the line between a good and a service, forming a continuum between these two extremes. For instance, a bellhop service in a hotel has a very different level of tangibility from a music streaming service such as the one provided by Spotify.

Other definitions focuses on lack of ownership, and the inherently instant nature of service provision, as well as clarifying the notion of a service→good continuum:

> [A service is] "*an act or performance offered by one party to another. Although the process may be tied to a physical product, the performance is essentially intangible and does not normally result in ownership of any of the factors or production.*" [Mehandjiev et al., 2010b].

Whilst this definition provides a helpful starting point in describing what we are interested in for the rest of this work, it is still too broad to be useful in the scope of this thesis. In particular, we need to focus this definition to describe exactly what we mean by a "software service", in order to describe the kinds of service that users of SC applications and EUSC applications would be able to use and create in the SC process.

Software Services

Software services move the generic definition of a service into the realm of computing, where the service that is provided is provided by a piece of software as opposed to an individual. One suggested definition for a software service is:

> [A software service is] *"a self-contained functional unit in which service consumers interact with the service through a well-defined interface."* [Obrenović and Gašević, 2008].

This definition identifies two important properties of services: that they are self-contained units (or black boxes), and that they provide a well-defined interface through which other entities can interact with them without knowing what is inside the service. A similar definition clarifies this point further:

> *[Software] "Services are self-describing, open components that support rapid, low-cost composition of distributed applications"* [Papazoglou and Georgakopoulos, 2003].

Papazoglou and Georgakopoulos's definition agrees with Obrenovic̀'s definition of software services, but goes on to include the concept of SC within the definition, identifying how SC is made possible by description of services. This well-defined, self-described interface must be provided to other entities through some kind of service description (for instance in the Web Services domain, the Web Service Description Language [WSDL] is used). Service descriptions are provided by services, and are used to advertise their capabilities, interface, behaviour and quality [Papazoglou and Georgakopoulos, 2003], both to other services and to users. The information provided in service descriptions is the information that can be used by other entities to allow discovery, composition, and execution of these services.

There are two types of service that are used within SC applications: component services (components) [Mennie and Pagurek, 2000] and composite services (composites) [Stecca and

Maresca, 2010].

Components are the services that are the input to the SC process, i.e. they are the services that the user composes together in order to create some new service. A component is a self-contained section of code with a well-defined interface, attributes, and behaviour [Mennie and Pagurek, 2000]. Components can be seen as black boxes that need to be coordinated with other components in order to provide useful functionality. For instance, if we consider the example scenario in the introduction, the component services are the tube lookup service, alarm service, and timer service.

Conversely, composites are the services that are the output of the process, i.e. they are the new service that is created by composing together components [Ibrahim et al., 2010]. Composites can be presented as a single entity (e.g. in Tasker[6]), or as a list of components (e.g. in If This Then That (IFTTT)[7]). These EUSC applications are discussed in Section 2.7 (page 25).

Neither of these definitions preclude a service from being both a component *and* a composite in different 'instances' or 'rounds' of composition. Figure 2-1 shows this concept in a graphical form. It relies on a sharing mechanism between users of the SC application, and allows a user to take a service that has been created by another user, and use this service as an input to their own composition. We refer to using composites as future components as *infinite composition*.

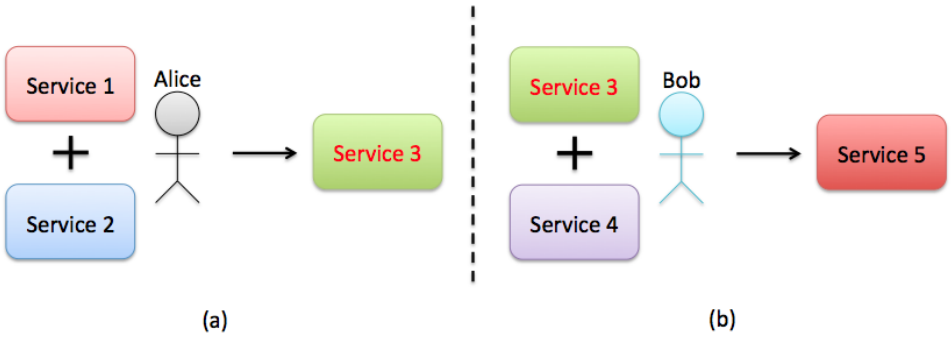


Figure 2-1: Infinite composition: (a) Alice creates a new service through SC [Service 3]. (b) Bob uses the service created by Alice [Service 3] to create a new service for himself.

However, SC applications do require a set of components that are *not* composites to initially populate the service 'pool'. These components cannot be decomposed any further, and will hereafter be referred to as *atomic components* (note that this term has been used in other work [Wajid et al., 2010], but it is unclear whether or not they are precluding infinite composition).

[6] https://play.google.com/store/apps/details?id=net.dinglisch.android.taskerm

[7] http://ifttt.com

In the rest of this section, we discuss components and composites in greater detail, as well as providing some clarification as to the information that these components and composites need to provide for them to be composable or executable.

2.3.2 Component Services

Components (component services) are the entities with which the user of the SC application has to work to create the new service that they require. Components require two things: an implementation that can be executed by some external entity, and a set of descriptive properties that can be used to discover them, either by other services or by users. These properties may include [Mennie and Pagurek, 2000]:

- A description of the component, including its functionality and constraints on its operation.
- The dependencies that the component has on other components, or on any supporting infrastructure.
- A list of the relationships that this component can form with others (i.e. other components that this component is compatible with)

Other definitions of components state that they are entirely autonomous, described uniformly across all compatible components, and that they are retrievable from some public repository via a given service discovery mechanism [Lizcano et al., 2008].

The descriptive properties of a component service are part of its contract (or advert), and are the only means by which the user of the SC application can determine what the component does, and whether it meets their needs for the composite service that they wish to make. This is an important part of the process of SC that has as yet not been evaluated from a user-perspective (service contracts are formally specified in Web Services, in machine understandable formats such as WSDL [W3C, 2001, Martin et al., 2004]).

Atomic Components 'Atomic service' is used without definition in several works [Rao and Su, 2005, Al-Sharawneh and Williams, 2009, Rodríguez-Mier et al., 2010], however it is clear from the context that their definitions of atomic service are similar to ours, if not identical. The main implication of infinite composition is that each time the user creates a composite, they are adding another component to the 'service pool' of candidate components that can be used in subsequent rounds of composition. It is important to distinguish between these components and the original set of components that were used to seed the pool. We define these 'atomic' components (or services) since they cannot be decomposed. Hence an atomic service must always be a component, but a component must not always be an atomic service.

2.3.3 Composite Services

Composites are the output of the SC process, and hence are created by composing a set of components [Ibrahim et al., 2010]. There are no stringent definitions as to the interfaces that these composites need to provide, what form they might take, or how they can be executed.

For EUSC applications that support infinite composition, there needs to be some sort of mechanism that allows the mapping of the service contracts of the components within the composite onto the composite itself in a way that adequately describes the composite to the EUSC application and to the user in such a way that it can be used in composition.

2.4 SC Life Cycle

In this section, we discuss two different views on task breakdowns of the EUSC process that identify the tasks that need to be completed in order for composition to occur. The first of these life cycles is the SC life cycle presented by [da Silva et al., 2008, da Silva et al., 2010], which describes the eight distinct stages of the SC process from atomic service creation through to composite execution, from the viewpoint of three different stakeholders: (atomic) service developer/provider, (composite) service developer, and end-user. This life cycle assumes an automated view of composition, where composites are created without direct intervention by the composite service developer. Silva et al.'s life cycle is listed below, and for each stage, we describe its purpose and identify the stakeholder who controls or is affected by that stage in brackets [da Silva et al., 2008, da Silva et al., 2010]:

1. **Service creation** (Atomic service developer/provider): The initial set of atomic services that are to be used in composition are created by a developer/provider, which is the only stage of the process in which developers are explicitly involved.
2. **Service publication** (Atomic service developer/provider): The atomic service that has been developed is published into a service repository from which they can later be discovered by other users.
3. **Service request** (Composite service developer): The composite service developer provides their requirements and preferences for the composite that they require.
4. **Service discovery** (Composite service developer): The system discovers components that could be composed with one another to create a composite that would meet the requirements in the service request. Discoverable components are contained within a service repository, and discovery is based on the services' contracts.
5. **Service composition** (Composite service developer): The system composes the services to meet the requirements in the request. Multiple compositions may be returned, from which the developer would select their preferred choice [da Silva et al., 2008]. This could be done automatically if the composite service developer provides a list of preferences for given properties of a service in the request phase.

6. **Service delivery** (Composite service developer): The composite that has been created is delivered to the composite service developer, and is prepared for execution.
7. **Service deployment** (End-user): The composite service that has been created and delivered is deployed to some composite repository where it can be discovered by end-users.
8. **Service execution** (End-user): The composite can be discovered from the composite repository by the end-user and can be executed.

A graphical representation of the stages of this life cycle and how they relate to one another and to the stakeholders are shown in Figure 2-2.

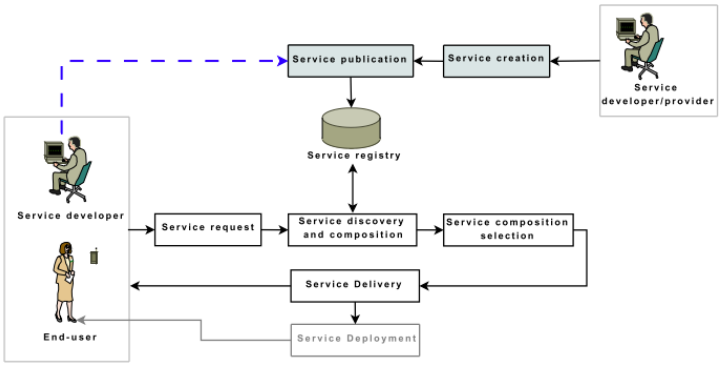


Figure 2-2: The Dynamic Service Composition life cycle (from [da Silva et al., 2008])

[Mehandjiev and De Angeli, 2012] present an alternative life cycle that is based on service-oriented software engineering rather than SC specifically. Their life cycle does not assign particular stakeholders to each stage, but instead discusses the prospect for end-user involvement at each juncture.

1. **Inception**: The initial definition of the project, where stakeholders assess the *feasibility* of the project, as well as undertaking *project planning*. End-user involvement at this stage can be due to their role as clients or owners of the project.
2. **Analysis**: The *domain* in which the Service-oriented application will operate, and the *activities* to be completed are analysed. If the end-user is the client or project owner, then they might be expected to provide knowledge that can be captured by the analysis process.
3. **Specification**: A specification is created detailing the properties that the service-oriented application needs is developed. Specifications record users' *needs*, *functional requirements*, and *QoS requirements*.
4. **Realisation**: This is the stage at which the service-oriented application is developed, and is itself made up of a number of possible sub-stages:
   (a) *Discovery*: Services that already exist and meet users' requirements are discovered, and presented to the user. (If this occurs, skip to provisioning).

(b) *Design*: If no services are discovered that meet the user's requirements, then a new one must be designed that does.
(c) *Construction*: The service designed in the previous stage is constructed through some composition process.
(d) *Verification*: The service-oriented application that has been constructed is verified to ensure that it operates as designed.
(e) *Validation*: The service-oriented application is validated against the requirements of the user.

5. **Provisioning**: Provisioning focuses on performing the activities required to enhance the composite so that it can be discovered and interacted with by other users of the SC application. The two stages of provisioning are *annotation* of the services with relevant information, and *deployment* to some repository where it can then be discovered.
6. **Management**: The stage at which the service-development process becomes more dynamic. *Monitoring* of the operation of the service-oriented application to ensure that it is operating within its expected parameters, and *adapting* the composite by returning to earlier stages (from Specification through to Provisioning) and modifying the composite.

The main weakness of [da Silva et al., 2008]'s life cycle is that it does not account for the dynamic aspect of the process (despite being called the dynamic service composition life cycle), where the user can adapt the composite that was created at the composition stage of the process. Furthermore, it focuses on an automated view of composition, where the SC application generates the composite automatically based on the requirements of the user that are provided in the specification stage. This means it is lacking in any discussion of user support through discovery and composition. This life cycle also shows a terminology problem, in that one of the stages is named "service composition", whilst at the same time being part of the "service composition life cycle"

[Mehandjiev and De Angeli, 2012]'s life cycle breaks the process into a much larger number of stages, with more consideration of the earlier stages of the process. There is no provision for the creation of atomic components, perhaps because of the focus on service-oriented software development rather than specifically SC.

Given these limitations, when we refer to the EUSC life cycle later in this work, we assume an amalgamation of these two life cycles: one which does not assume automated composition, facilitates dynamic modification of the composite, and explicitly considers the creation of atomic component services. This life cycle is presented later in the thesis.

2.4.1 SC Dynamicity

The 'dynamicity' of the SC process is related to the SC life cycles described above since dynamicity discusses whether the process occurs at design-time or run-time. It is normally

considered that SC is static if composition is performed in a separate design phase, or dynamic if composition is performed at run-time [Zhang et al., 2003, Laga and Bertin, 2012, da Silva et al., 2009, Mennie and Pagurek, 2000].

[Laga and Bertin, 2012] identify SC as a process that is always completed at design-time. That is, users create a composite at design-time, the composite is executed at run-time, and to implement any changes that need to be made, the user must return to design-time. They criticise this approach for lacking in dynamicity, although it does consider changes in user requirements unlike the life cycle suggested by [da Silva et al., 2008].

We argue that the management stage of [Mehandjiev and De Angeli, 2012]'s life cycle provides the dynamicity of the process since it allows the composer to modify the composites that they have created, regardless of whether it is classed as design-time or run-time.

2.5 Stakeholders of SC

To properly understand EUSC, we need to consider the stakeholders who are involved in the various stages of the process. We will split our discussion of stakeholders into two sections: the roles that need to be fulfilled, and the actors who can fulfil those roles. This section also helps us to provide some context as to who the 'End-user' is in End-user Service Composition.

2.5.1 Roles

The roles in SC are determined by the different tasks that need to be performed. [da Silva et al., 2008] define four roles that need to be fulfilled in the SC process.

- **Service developer:** The creator of the software service.
- **Service provider:** The provider of the software service.
- **Composer:** The actor performing composition.
- **End-user:** The actor executing the service that has been created via composition.

We feel that development and provision of the components to be composed can be generalised into a single role. Figure 2-2 shows these roles with the service developer and provider amalgamated into a single role, implying that [da Silva et al., 2008] also felt that these roles could be carried out by a single individual or organisation.

If we consider the related sub-field of End-user Development (EUD) known as component-based tailorability, we find three similar activities that are identified, with corresponding roles that need to be fulfilled [Stevens et al., 2006]:

- **Designing new components:** The initial set of components to be used need to be created.

- **Configuring components:** Components can be configured by changing their parameters, before using them in any composition-like process.
- **Configuring the composition of components:** Components are coordinated through composition.

Considering the different roles that are identified above, we suggest the following set of roles as the three main constituents of SC:

1. **Component creator**: Creates the components that are used in composition.
2. **Composer**: Composes components or composites with one another in order to make new composites.
3. **End-User**: Uses the composites that have been created by the composer.

In EUSC, we assume that the user who fulfils the composer role is the same user who fulfils the end-user role. No assumptions are made as to the identify of the component creator, who could be the creator of the EUSC application, a different application developer, or even the end-user.

2.5.2 Actors

Now that we have identified the roles that need to be fulfilled in the composition process, we must consider who fulfils these roles, which we define as 'actors'. These actors will be discussed in terms of the skills and knowledge that we assume they possess.

[da Silva et al., 2009, da Silva et al., 2010] distinguish between three different types of knowledge that are important for actors in SC:

- **Domain knowledge:** their knowledge of, or familiarity/experience with the domain in which the service they seek, or the components therein operates.
- **Technical knowledge:** their understanding of technical concepts that underpin SC.
- **Service knowledge:** their knowledge and awareness of the concepts relating to services and service-oriented computing.

The above knowledge classifications prompt [da Silva et al., 2009, da Silva et al., 2010] to identify four different types of actor who can undertake one of the roles in composition: Layman; Domain expert; Technical expert; and Advanced user. These types of actor differ based on the amount of technical and domain knowledge they are assumed to have, as shown in Table 2.1. Note that Table 2.1 does not explicitly consider service knowledge since it is a subset of technical knowledge given that users without technical experience would have very little knowledge of SC or SOC [Namoun et al., 2010c, Namoun et al., 2010a, De Angeli and Battocchi, 2011].

[Nestler et al., 2011] identify two further types of user, based on a blend of their knowledge and their day-to-day activities: information workers and everyday Web users. Information workers are office-based employees who interact with standard software packages such as

Table 2.1: Table of user types (from [da Silva et al., 2010])

User type	User knowledge		
	Domain	**Services**	**Technical**
Layman	No	No	No
Domain expert	Yes	No	No
Technical expert	No	No	Yes
Advanced user	Yes	Yes	Yes

MS Office (particularly information packages such as MS Excel). Everyday Web users have experience with blogs, wikis, or social networking services, although would not use these in their job. We argue that the main difference in this case is the context in which the actor is using the software package – particularly if they are being paid to do so.

The field of EUD suggests that the roles that users take can change over time. This is presented as the consumer-designer spectrum, identified by [Fischer and Giaccardi, 2006]. Figure 2-3 presents the different roles that can be taken by consumers or designers within EUD, showing the side of the spectrum to which each role leans. [Fischer and Giaccardi, 2006] assert that actors can migrate along this spectrum over time, meaning that through sustained use of EUD applications, they would be able to migrate from being a passive consumer, through to power user, and eventually meta-designer.

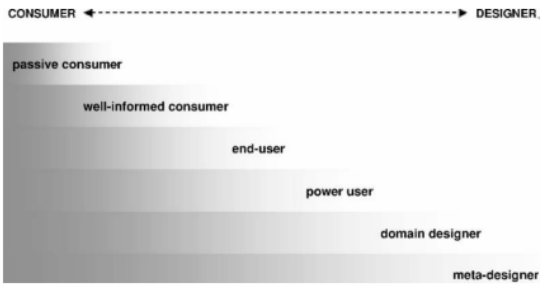


Figure 2-3: The Consumer-Designer Spectrum (from [Fischer and Giaccardi, 2006]).

This idea of migration through the consumer-designer spectrum can also be applied to our work, in that initially an end-user may only consider being the final user of composites created by others, but over time they may become a composer, and eventually even a component creator.

Now that we have considered the types of user that can take part in composition and the roles that need to be fulfilled, we are able to turn our discussion to EUSC.

2.6 End-user Service Composition

The definition of End-user Service Composition (EUSC) has two salient points:

1. **The user who performs the composition step is also the user who executes the composite that is created in this process.** This precludes situations where users in one department of an organisation compose a service to be used by other users in the organisation. It is worth noting that this definition should not preclude the possibility of sharing composites with other users of an EUSC application. Hence, we extend the notion of the user to some class of user.
2. **No assumptions are made about the level of technical proficiency of the user performing composition.** Since we cannot make any assumptions about the level of technical skill of the user, we must account for users who have very low technical skills as well as those with very high technical skill. However, We can assume a certain level of domain knowledge since they would require some level of domain knowledge in order to be able to understand the composite to be created. This definition is similar to [Obrenović and Gašević, 2008], who define EUSC as SC where the composer is not a professional developer, and their day-to-day job is not within Computer Science.

[Namoun et al., 2010d] performed a study to assess their participants' opinions on SC, and found that participants showed a high level of interest in SC, and they felt that it would be useful to them. They did however acknowledge that it would be a relatively error-prone process. [De Angeli and Battocchi, 2011] found that participants felt that SC could be a process that would allow them to automate repetitive tasks in their day-to-day lives.

A number of positive ideas emerged from the work of [Namoun et al., 2010d]:

- Users can "Go with the flow": if a user's friends are performing SC, then they are more likely to join in too.
- Examples are required to demonstrate useful, non-trivial compositions that the user can perform.
- A community should be associated with the EUSC application to allow users to share their creations with one another, which can be discovered or rated by other users.

Figure 2-4 shows the "long tail" of user requirements [Picozzi, 2014], where popular applications (Region A in Figure 2-4) are created by application developers, and those which are less popular (Region B) – the long tail – is an area where EUD, EUSC and Mashups can be utilised by users to create the applications they require themselves. This motivation is reflected by the rise in popularity of available EUSC applications. We will describe the EUSC applications that are used in this thesis in Section 2.7.

[Casati, 2011] notes that current EUSC applications have three main aims: to be **generic**, **powerful**, and **simple**. They assert that the problem with these aims is that all three cannot be achieved at once, and that at least one of these aims must be dropped in order to achieve

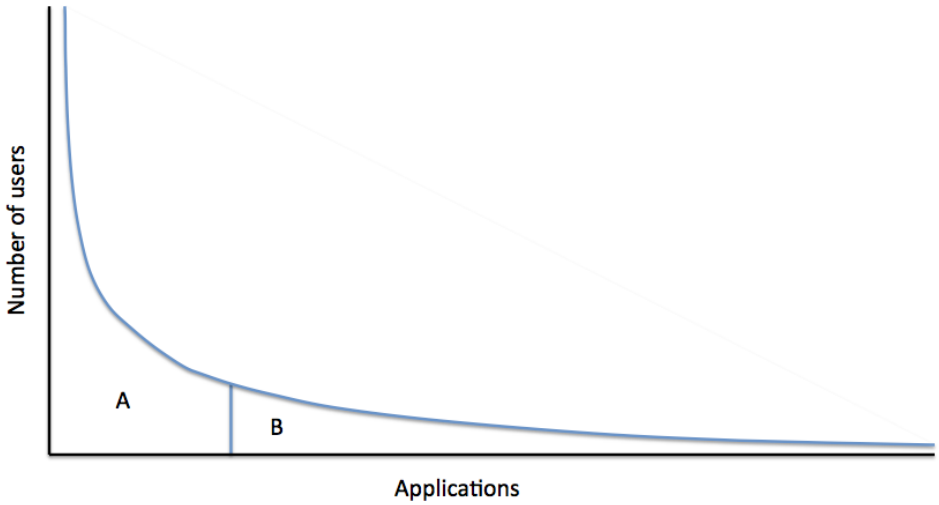


Figure 2-4: The long tail of application requirements

one of the other two [Casati, 2011]. Casati states that the aim of being generic is the aim that should be dropped, and is indeed the case in the currently popular applications such as If This Then That (IFTTT), which focuses on Web services. IFTTT is described in detail in Section 2.7.

In [Namoun et al., 2010a]'s study aiming to identify the perceptions of end-users performing SC, they found problems including a low awareness of SC, no understanding of control or data flow, through to worries about security .

2.6.1 Mashups

Later in this chapter, we will see that there are a number of approaches to creating design spaces within the EUSC domain, all of which target mashups. Mashups are a specialised class of EUSC application that are based on the Web. Consider these two definitions for 'mashup' in a computing context:

- Merriam-Webster: *"a Web service or application that integrates data and functionalities from various online sources"* [8].
- Oxford Dictionaries: *"A web page or application created by combining data or functionality from different sources"* [9].

These definition show the parallels between Mashups and Service Composition in that the definitions are all variations on the theme of connecting data from different sources on the Web. It has been commented that it is difficult to understand exactly what a mashup *is*, and

[8] http://www.merriam-webster.com/dictionary/mash-up
[9] http://www.oxforddictionaries.com/definition/english/mash-up

what it *is not* [Yu et al., 2008]. In particular, we suggest that mashups relate to SC as EUD of mashups relates to EUSC.

Given the similarities between SC and mashups, we will first define the terminology that we will use when discussing mashups, and relate these terms to those that we have used when discussing SC:

- **Mashing up**: The process of combining various Web data sources or processes; equivalent to *Service Composition*.
- **Mashups**: The output created in the mashing up process; equivalent to *Composites*.
- **Mashup Development Environment/Mashup Development Application**: The tool or application that is used to perform mashing up and create mashups; equivalent to *EUSC applications*.

Mashups are Web applications that combine online resources, often from a number of third parties [Koschmider et al., 2009, Fischer et al., 2009, Yu et al., 2008]. As with SC, mashups can be categorised as data mashups (combining a number of data sources), process mashups (connecting together a number of processes) or presentation mashups (combining together a series of user interfaces) [Koschmider et al., 2009, Grammel and Storey, 2010]. Mashups have been identified as supporting both consumers and enterprise users [Na et al., 2010].

Over the past 10 years, a lot of Web-based EUSC or Mashup Development applications targeted at end-users have failed [Casati, 2011], including: Microsoft Popfly[10], QEDWiki/IBM Lotus Mashups[11], Intel Mash Maker[12], Google Mashup Editor[13] to a myriad of projects described in academic papers, potentially due to their technology-driven approach [Mehandjiev et al., 2014]. EUSC applications have become popular again more recently on mobile platforms, notably on Android. We suspect the current increase in popularity is due to the availability of Web services and Web APIs that the developers of these applications have access to, as well as the vast functionality available on mobile devices.

2.6.2 End-User Development (EUD)

EUSC falls into the domain of End-user development (EUD). The main aim of EUD is to allow end-users to develop and adapt the systems and applications that they require, to remove their reliance on professional developers [Lieberman et al., 2006, Danado and Paternò, 2012]. EUD is defined as:

> *"EUD can be defined as a set of methods, techniques, and applications that allow users of software systems, who are acting as non-professional software de-*

[10] http://en.wikipedia.org/wiki/Microsoft_Popfly
[11] http://www.ibm.com/developerworks/lotus/products/mashups/
[12] http://software.intel.com/en-us/articles/intel-mash-maker-mashups-for-the-masses
[13] https://developers.google.com/mashup-editor/

> *velopers, at some point to create, modify or extend a software artefact."* [Lieberman et al., 2006]

One of the biggest problems in EUD can be summarised by the Turing tar pit, and its inverse:

- Turing tar pit:
  > *"Beware the Turing Tar Pit, in which everything is possible but nothing of interest is easy."* – Alan Perlis [Perlis, 1982]
- Inverse Turing tar pit:
  > *"Beware of over-specialised systems, where operations are easy, but little of interest is possible"* [Fischer and Giaccardi, 2006]

The trade-off between the Turing tar pit and its inverse is identified as the trade-off between simplicity and expressive power, which is reflected in the trade-off identified by [Casati, 2011] for EUSC: power, simplicity, and how generic the EUD application is. Suggested methods for balancing this trade-off are to simplify the EUD process [De Angeli and Battocchi, 2011], re-use knowledge between users [De Angeli and Battocchi, 2011] to reduce the load on a particular user, and provide a large amount of assistance to users [Cappiello et al., 2011b].

A further trade-off in end-user development is balancing the cognitive load of programming and the benefit that the user obtains from performing the activity, which is a fundamental component of the Attention Investment Model [Blackwell, 2002]. Another framework that is meant to support EUD is meta-design [Fischer and Giaccardi, 2006], where user become a co-designer throughout the lifetime of the system, and is able to extend the system to meet their needs. Clearly EUSC falls within this definition.

2.7 EUSC Applications

In this section, we provide a brief overview of some of the commercially available EUSC applications that are used as part of this thesis. We pay particular attention to those that were used in our studies to demonstrate EUSC to our participants. Other applications have since become available, and were used as part of our design space creation method. In this discussion, EUSC applications are grouped by the platform on which the composition portion of the application operates.

2.7.1 Web-based EUSC Applications

We identified three EUSC applications that operate on the Web:

- If This Then That (IFTTT): `http://ifttt.com`
- Yahoo! Pipes: `http://pipes.yahoo.com`
- On{X}: `http://onx.ms`

IFTTT was first released as a Web application, before mobile versions were created for both iOS and Android. The premised upon which IFTTT is based is shown in Figure 2-5.

Figure 2-5: An explanation of IFTTT (from `http://ifttt.com/wtf`).

IFTTT provides uses with a mechanism for creating compositions that fit the template: if [Trigger] then [Action], which is a commonly used template in modern EUSC approaches, presumably due to its simplicity. IFTTT provides components through channels (grouped into categories, normally by the provider of the service). As of August 2014, there were 124 channels provided through IFTTT, spread across many Web services (e.g. Facebook, Twitter, GitHub), pervasive services (Nest thermostat, Philips Hue, SmartThings), and mobile only functions (Android Notifications, SMS, etc.). Composition is performed on IFTTT by the user 'filling in' the base template by first selecting and configuring a trigger, before subsequently selecting and configuring an action.

Yahoo! Pipes is a mashup tool that allows users to perform a number of different actions related to RSS feeds. They can combine and filter these feeds using a number of different components that are provided within Yahoo! Pipes. Visually, Yahoo! Pipes is totally different to IFTTT in that the user has a canvas upon which they can position the components and connect them using 'wires'. A sample mashup in Yahoo! Pipes is shown in Figure 2-6.

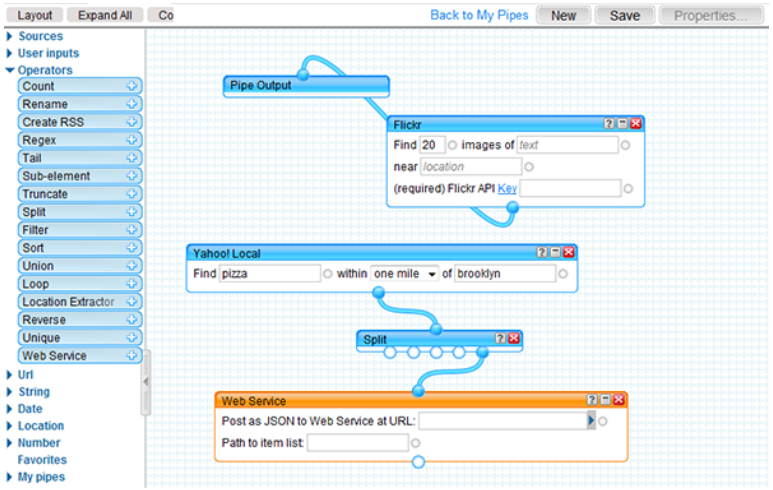


Figure 2-6: A sample mashup in Yahoo! Pipes

On{X} is a much more complex EUSC application that relies on composition being performed using a domain-specific programming language based on JavaScript. Composites are created in a text editor online, and the resulting composite application is pushed to a mobile

application, upon which the composite it executed. As such, components in On{X} all relate to functions provided by mobile devices. The composition process in On{X} is shown in Figure 2-7.

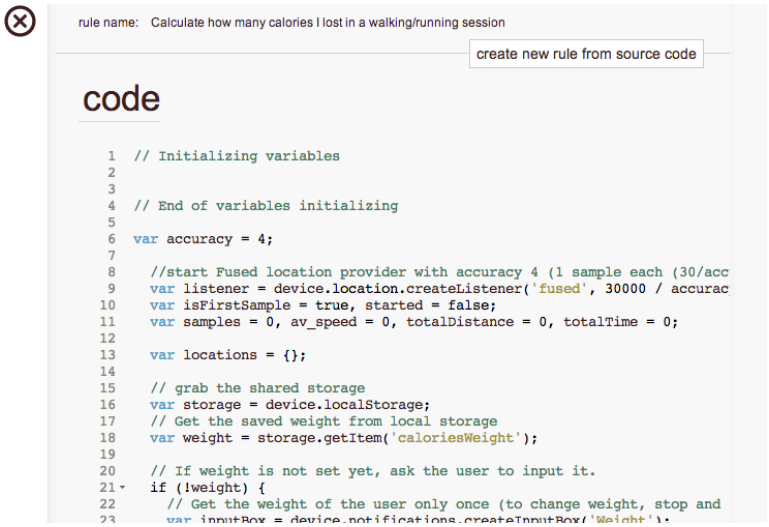


Figure 2-7: A sample composition in On{X}

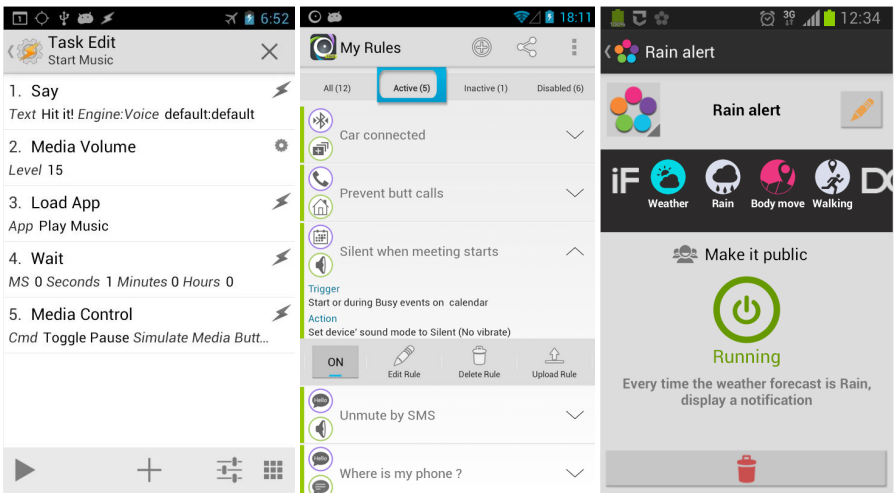
2.7.2 Mobile EUSC Applications

Three available mobile EUSC applications were provided to participants in the studies described in this thesis:

- Tasker: `https://play.google.com/store/apps/details?id=net.dinglisch.android.taskerm`
- AutomateIt: `https://play.google.com/store/apps/details?id=AutomateIt.mainPackage`
- Atooma: `https://play.google.com/store/apps/details?id=com.atooma`

Tasker is the oldest of the Android-based EUSC applications, and currently the most complex. Users can select from a number of mobile-oriented components and position them in a linear composition. These can be activated by an external trigger, at a particular time, or exported as an application that the user can execute themselves. The main difference between Tasker and other such applications is that Tasker does not support passing of data between components. A sample task in Tasker is shown in Figure 2-8(a).

AutomateIt is another Android-based EUSC application, which operates using the same principle as IFTTT: the execution of an action when trigger conditions are met. The components that are available are similar to those on Tasker, in that they have a mobile focus.



(a) Tasker (b) AutomateIt (c) Atooma

Figure 2-8: Sample compositions in each of the mobile EUSC applications.

Furthermore, components do not pass data between each other. A list of example composites in AutomateIt are shown in Figure 2-8(b).

Atooma takes the simple template that is used by IFTTT and AutomateIt and extends it to include boolean logic: if this AND this then that AND that. The components are grouped into categories based on where they are based – either mobile or Web services, before being grouped into more specific categories. Atooma supports the passing of data between components. An example composite in Atooma is shown in Figure 2-8(c).

Other mobile EUSC applications that were not presented to participants in these studies were: IFTTT (Android), IFTTT (iOS), Automated Device, E-robot, Condi.

2.7.3 Desktop EUSC Applications

Two desktop-based EUSC applications were presented to participants as part of the studies documented in this thesis, both of which are available within Mac OSX:

- Automator
- Quartz Composer

Automator is a desktop automation application that connects together functions within applications provided in OSX, which can be connected together linearly. The components that are provided in Automator are grouped based on the application that provides them, and relate both to features built in to OSX, or provided by other applications such as Microsoft Office. A sample composition in Automator is shown in Figure 2-9.

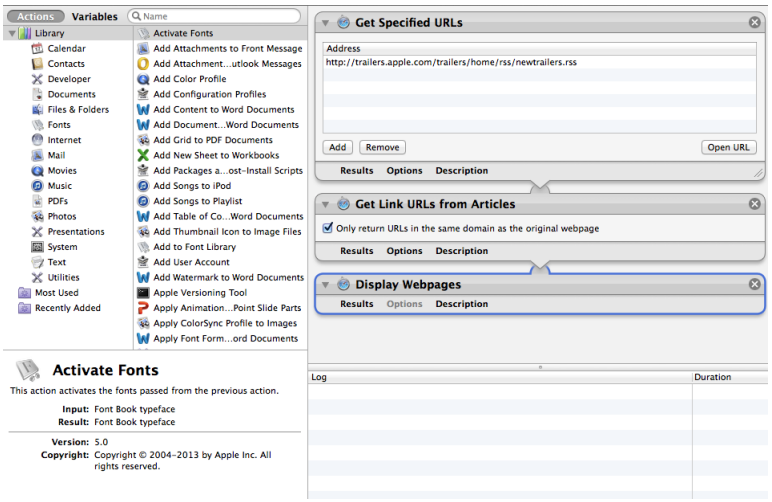


Figure 2-9: A sample composition in Automator.

Quartz Composer allows users to create multimedia compositions based on wiring together multimedia-related components, in a similar manner to Yahoo! Pipes. Components in Quartz Composer present a list of inputs and outputs that the user can either connect to other components, or set input values. Compositions in Quartz Composer do not present the order in which components are executed, instead the wires represent the flow of data. A sample composition in Quartz Composer is shown in Figure 2-10.

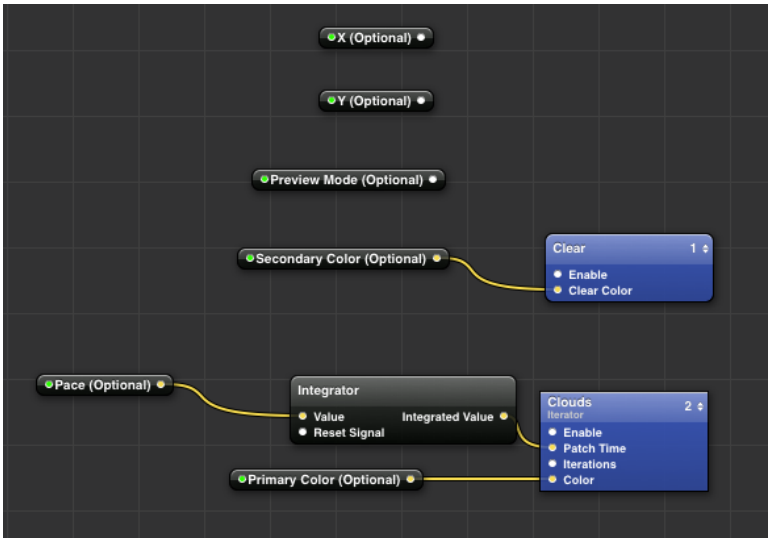


Figure 2-10: A sample composition in Quartz Composer.

2.8 Requirements Gathering for EUSC applications

In the introduction to this thesis, we identified one of our research goals as being to derive a comprehensive set of requirements for EUSC applications, and thus needed to find requirements for EUSC applications that have already been derived. We were only able to identify two pieces of prior literature that reported gathering of requirements for EUSC applications. Both used focus groups to try to gather requirements following the main themes of SC, and to assess how familiar their participants were with these themes. [Namoun et al., 2010b] focused on requirements for EUSC applications on the Web, whereas [Mehandjiev et al., 2010b] were interested in their participants' perceptions of, and requirements for, EUSC.

[Namoun et al., 2010b]'s focus groups sought to identify users' knowledge and perceptions of software services, service-oriented computing and SC to motivate a set of requirements for a proposed EUSC application. Their focus group sessions contained six steps:

1. Participants were provided with a list of terms that are commonly used within Service-oriented computing (SOC), and they were asked to provide definitions of these terms to assess their knowledge of the domain. Following this, they were provided with the correct definition for each term.
2. An initial mock-up of the design of the proposed EUSC application was presented to participants, and they were asked to comment on the design.
3. Participants were guided through a sample composition using the proposed design following a scripted sequence.
4. Participants were asked for their opinions on EUSC in general, before being presented with alternative design mock-ups for the proposed EUSC application.
5. A prototype of the proposed EUSC application was demonstrated by the researchers, the design of which matched the initial mock-up that was shown to participants earlier.
6. Participants were asked to provide opinions on EUC in general for a second time, as well as being asked to provide further detail into the approach that was taken in the prototype.

After an analysis of the data that was gathered in their focus group sessions, [Namoun et al., 2010b] present the following set of requirements:

R1. Represent components in the composition with their normal user interface.
Component services should be presented in a WYSIWYG manner, i.e. services should be represented with the same interface during the composition process as the one that the user is used to if they are using the service directly.

R2. The application should use a semi-automated approach to composition.
The application should assist the user with certain aspects of the process by performing them automatically, for instance automatic matching of inputs and outputs of components.

R3. Avoid technical terminology or jargon.

Terminology should be accessible and understood by users without technical or domain knowledge.

R4. The "canvas" of the composition should be large.
Users should be able to interact with the composition on a large canvas that they can navigate around easily.

R5. Components should be secure.
Components that require the user to enter personal information or passwords should store this information securely.

R6. Users should be provided with proactive feedback throughout the process.
Feedback should be provided to the user of the application without them having to seek it out.

[Mehandjiev et al., 2010b] also carried out a focus group study, but the aim of their focus groups was to assess participants' perceptions of the flow between the components in the composition process, rather than EUSC in general. 'flow' in an SC context normally either refers to control flow (i.e. the order in which components are executed), or data flow (the data that is passed between components in composition).

[Mehandjiev et al., 2010b]'s focus groups were made up of 5 steps:

1. Participants were given various EUSC-related terms to which they were asked to provide definitions. Example terms were *service* and *software service*.
2. Participants were given a 20 minute introduction to give them information on SC and other related concepts.
3. Participants were given a questionnaire to assess their technical skills or knowledge, and to determine if they had any other experience in Service-oriented Computing.
4. Participants were asked to perform three EUSC tasks to determine how they would complete the process if left to their own devices.
5. Finally, participants were given a presentation to give them insight into the different possibilities within SC. Alongside this presentation, they were given a questionnaire to assess their opinions on the different alternatives for representing flow in SC – particularly control vs. data vs. a bespoke "assisted composition" approach in which no flow was presented.

The output of the questionnaires given to participants at the end of the sessions were analysed, and the following set of requirements generated [Mehandjiev et al., 2010b]:

R7. Data flow should not be represented.
The data that is being passed between the components in composition should not be represented explicitly to the user.

R8. Sets of data should be ignored.
If data in composition is a set of data, then it should be represented as a single element of that set instead. For example, if a component returned a list of contacts, the component should only indicate that it returns a 'contact' and iteration should occur in

the background.

R9. Assistance with control flow is required.
The application should provide assistance to the user to solve any control flow dependencies or problems that they might encounter whilst performing composition.

R10. Assistance with data flow is required.
The application should provide assistance to the user to solve any data dependencies or data flow problems that they might encounter whilst performing composition.

R11. Components should be represented in a WYSIWYG.
The components in the composition should be represented with the interface that they would present if the user were interacting with the service itself outside of composition.

This small set of requirements contains some conflicts: notably between **R7** and **R8**. That is, **R7** states that data flow should not be represented at all, and **R8** discusses how data should be represented, which is a clear contradiction.

The above sets of requirements are both small, and no guidance is provided as to how these requirements were elicited from the data that was gathered as part of the focus groups. Furthermore, a similar method was used in both approaches. These requirements provide two slightly differing viewpoints on requirements for EUSC: [Mehandjiev et al., 2010b] focused on flow – a single aspect of EUSC, and [Namoun et al., 2010b] had a more general approach.

The requirements presented here give us a strong motivation for further investigation into potential requirements for EUSC, not least because the specialised set of requirements investigating one area of SC [Mehandjiev et al., 2010b] generated nearly as many requirements as the more general approach [Namoun et al., 2010b]. Furthermore, we believe that a methodological contribution is possible, since neither of the reviewed requirements gathering approaches gave any detail as to how the requirements that they gathered were elicited from the data that was gathered as part of the focus group sessions.

2.9 Design Spaces

Our work intersects EUSC and design spaces, and so far in this literature review we have discussed the scope of our work on EUSC, and now we need to do the same with design spaces.

The term ‘design space’ is used throughout design literature, and is frequently used in HCI without being defined (consider various CHI 2012 submissions across a number of different topics within HCI [Szafir and Mutlu, 2012, Kriplean et al., 2012, Pierce, 2012, Parker et al., 2012]). Design spaces were first suggested by Lane as a tool to identify functional and structural design decisions, classifying the various solutions for each decision [Lane, 1990, Lane, 1996]. Since then, however, design spaces have been used in various different

Table 2.2: Definitions of "Design Space"

Author	Definition
[Lane, 1990]	*"A design space identifies the key functional and structural choices made in creating a system design, and it classifies the alternatives available for each choice."*
[Lane, 1990]	*"A multi-dimensional design space that classifies system architectures. Each dimension of a design space describes variation in one system characteristic or design choice. Values along a dimension correspond to alternative requirements or design choices."*
[MacLean and McKerlie, 1995]	*"This design space is an explicit representation of alternative design options and reasons for choosing among those options."*
[Baum et al., 2000]	*"A design space is a multidimensional space of design choices. It is spanned by a set of* dimensions *identifying relevant criteria for characterizing artefacts in a specific domain."*
[Wood and Agogino, 2005]	*"A typical design space model maps the set of design variables (aspects of the design under the direct control of the designer) onto an aspect of performance of interest to the designer."*
[Westerlund, 2005]	*"The model uses the 'design space' as a conceptual tool that can be used both for designing and understanding design processes. The design space is here understood as all the possible design solutions. In reality the design space is an extremely complex multi-dimensional space containing an endless amount of solutions, but we are here only interested in it as a concept."*

ways warranting further investigation into *what* they are and *how* they are used. Other definitions of "design space" are shown in Table 2.2.

Table 2.2 shows that the base definitions of the concept of a design space are broadly in agreement across those authors who have sought to provide a definition: a multi-dimensional space made up of design decisions and solutions presented as solutions to those decisions.

One of our research goals is to create a design space for EUSC applications. To do this, we must first describe exactly what we mean by 'design space'. In the rest of this section, we assume that a design space is a multi-dimensional space that presents design decisions across dimensions, with possible solutions to these decisions as points on each dimension. Our discussion of design spaces in this section is focused on what design spaces can contain, and what they can be used for.

2.9.1 Design Space Contents

Design spaces are defined as being made up of a set of *dimensions*, each of which present a single design choice, and thus a method of characterising an aspect of a design artefact in a particular domain based on its 'position' within the design space [Lane, 1990, Baum et al., 1998, Geyer, 2000]. These design choices can operate at different levels of abstraction, either at a high level such as the work by [Mehandjiev and De Angeli, 2012], whose dimensions were stages of the EUSC life cycle, and aspects of End-user Development (EUD) that could be supported. Alternatively, decisions can be at a much lower level, such as [Pietschmann et al., 2010], who discusses the specific technologies involved in the design of components in Mashup Development Environments. Since we are creating a design space in the EUSC domain, it is important that design spaces support the different topics that we have identified in EUSC literature. Regardless of the level of abstraction considered, these dimensions are still presented as decisions, and potential solutions.

Dimensions within design spaces are said to represent either questions [MacLean et al., 1991, MacLean and McKerlie, 1995] or design decisions [Westerlund, 2005, Baum et al., 2000]. The difference between these two views is the way that they are presented: questions are presented in the form of a question, e.g. "On what platform should the the application operate?". Design decisions on the other hand tend to be presented using a single term, e.g. "Platform". Both the question-based representation and the decision-based representation would present solutions in the same way (excluding design criteria) – in the case of the above example regarding the choice of platform, potential solutions (or options) might be "Mobile", "Desktop", or "Web". It is important to note that rather than all solutions being candidates for a single design, these potential solutions are a set of what *could* be chosen [MacLean et al., 1991]. In this thesis, we identify dimensions as design decisions rather than questions.

Dimensions of design spaces are suggested as having a number of properties based on the type of solution that is available to solve that design decision: dimension type and continuity, popularity and whether the dimension is mandatory.

Dimension types are based on what the design dimension represents in the design: classification variables, or performance variables. Classification variables are potential solutions to a design decision that are members of a set (such as the above platform choices), whereas performance variables are ordinal values such as a unit of measurement [Wood and Agogino, 2005]. Dimension continuity presents the solutions to dimensions as either being continuous or discrete. Continuity of dimensions is closely related to variable type, in that these ordinal performance variables can either be continuous or discrete, and that classification variables are discrete by definition.

The concept of a dimension being mandatory (or not) is a concept borrowed from a low-level form of design space known as Feature-oriented Design Analysis. That is, features can either be mandatory, optional, or alternative [Kang et al., 1990, Geyer, 2000]. Mandatory features

are features that must be chosen for a given type of system if their parent feature is chosen (analogous to a requires relation [Geyer, 2000], and necessity [Gooch, 2013]), optional features are those that *may* be chosen if their parent is but do not have to be (analogous to sufficiency [Gooch, 2013]), and alternative features are mutually exclusive alternatives for a particular decision [Kang et al., 1990, Geyer, 2000] (analogous to or-features [Czarnecki and Eisenecker, 1999]). [Geyer, 2000] generalises this concept slightly and presents common features (present in all systems in a given domain), and variable features (present only in some systems within the domain). A final related concept is excludes-relationships, where choosing one solution means that another must not be chosen [Geyer, 2000].

Whilst these properties of design decisions and dimensions are important, we do not believe they are fundamental to design spaces in the same way as other properties of the space, such as their structure and the contents of the design decisions and solutions themselves.

2.9.2 Design Space Structure and Size

We know that design spaces are collections of dimensions that represent design decisions to be made, and points on these dimensions represent potential solutions to that design decision. The next consideration we need to make before we can create our own design space is the relations between these dimensions, and how they should be structured.

Design space models are normally presented as taxonomies of design decisions [Geyer, 2000]. They are often represented in a tree-based hierarchy (e.g. Feature-oriented Design Analysis [Kang et al., 1990], [Geyer, 2000]'s design space). When used to evaluate domain applications, design spaces are more often shown in a tabular form in order to represent both the elements of the design space, and their inclusion within the tool(s) being evaluated (e.g. [Pietschmann et al., 2010, Minhas et al., 2012]).

Models of design spaces are typically large, and can have complicated links between dimensions – [Baum et al., 2000] has recorded sizes of between 40 and 80 dimensions. [Geyer, 2000] suggests that a remedy for the problem with design space size is to split design space models up into sub-models, which aligns with the functional and structural design spaces initially postulated by [Lane, 1990]. Within the sub-models of the design space, dimensions/decisions can also be grouped together with other similar dimensions in order to present dimensions covering similar topics near one another in the space [Geyer, 2000].

The large size of design spaces is also cited as a reason for software tools to be created to help designers interact with design spaces more effectively [Baum et al., 2000]. Tools can provide a number of supporting features, depending on how the design space is being used. We discuss tool support for design spaces in Chapter 5.

2.9.3 Design Space Usage

[Gooch, 2013] suggests two different uses for design spaces: *Generative* and *Evaluative*, both of which require a value judgement to be assigned to the entities within the design space (notably potential solutions), and differ only in the way in which this value is used. Further discussion as to the origin of this value is required, as well as more clarification about the relationship between the generative and evaluative design spaces. [Gooch, 2013]'s other type of design space is conceptual, and as such should not be considered as something that can be used in the same way as the other two types of design space.

[Gooch, 2013]'s definitions for these two types of design space are as follows: *"Generative design spaces assist designers in producing designs which are 'good' for some measure of good. Generative spaces seek to predict what design elements and characteristics will produce a positive outcome."* [Gooch, 2013]. *"Evaluative design spaces predict the outcome that a given design would produce."* [Gooch, 2013]. [Lane, 1990] provides further insight into the design rules to suggest that they could be formulated within a design space to show which combinations of choices are 'good' or 'bad', and thus can then be used to 'select' an appropriate system from within the design space.

Using design spaces in a generative manner is based upon generative design theory. [Jul, 2002] observes that designers can spend a large amount of time "re-inventing the wheel", or even "failing to recognise that a wheel would solve the problem". Generative design theory seeks to address this by providing a mechanism for designers to map scientific knowledge with existing designs and to better facilitate the generation of alternative designs [Jul, 2002]. Design spaces can help to provide some of the insight required in this generative approach, and to help produce designs that are 'good' [Gooch, 2013].

The use of design spaces in the evaluative sense means that designers can use value judgements to assess the success of a particular suggested design [Gooch, 2013], or to assess designs that are currently considered as the state-of-the-art in the domain [Gooch, 2013]. The examples of design spaces applied to EUSC applications (Section 2.13) show design spaces used in an evaluative manner, although without an explicit value judgement associated with each design choice.

Before design spaces can be used either evaluatively or generatively, some calculation is needed to determine the value of a particular solution. One mechanism for determining this value is calculating the correlation between particular design choices [Geyer, 2000]. [Geyer, 2000] suggests two kinds of correlation that can be calculated or presented: *strong* and *weak*. *Strong correlation* is a boolean value, indicating that if one solution is chosen, then another must be, or must not be. *Weak correlation* is a numeric value from -1 to 1 indicating the mathematical correlation between the two values. However, we feel it necessary to highlight that a strongly correlated pair of design elements would be equivalent to being 1 or -1 as a weak correlation. These correlations can either by *symmetrical* ($\alpha \Leftrightarrow \beta$) or *asymmetrical*

$(\alpha \Rightarrow \beta)$.

If we use correlation as a method for determining value for particular design choices, there needs to be a mechanism for identifying particular design choices within artefacts that already exist in the domain. This process is known as design space profiling [Westerlund, 2005] (or, alternatively, application profiling – we use this term to prevent terminology confusion). Once a number of existing applications have been profiled, it is then possible to calculate correlations between decisions or solutions within the design space.

[Aghaee et al., 2012] identifies three potential uses for the results of application profiling: to assess the evolution of decisions that have been made across the range of years in which the applications were released; to identify and evaluate the design decisions that are commonly chosen together or are chosen least frequently; and to assess the impact of choosing one design element on other design elements within the design space.

The final type of design space suggested by [Gooch, 2013] is the conceptual design space. Conceptual design spaces operate without any value judgements, instead describing only the concepts that exist within the space. [Westerlund, 2005] described conceptual design spaces as containing "all possible solutions", which clearly shows that they operate at a different level of abstraction to the more concrete evaluative and generative spaces. We believe that this difference in abstraction is understated in the original work, and Chapter 4 discusses how we feel that these definitions could be adapted to better reflect the differences between conceptual design spaces and both generative and evaluative design spaces.

One of the other uses of design spaces is to profile existing tools in the domain to identify the design choices that are made in these tools and thus find their position in the space. This process can also be used to evaluate conceptual design spaces, and is discussed further in Section 2.9.5.

2.9.4 Design Space Creation

As we have stated in previous sections, one of our research aims is the creation of an explicit design space for EUSC applications, as as such we must consider how other authors have approached the process of creating design spaces.

The first task that must be undertaken is the creation of a design space. This process is defined as Design Space Analysis [MacLean et al., 1991], and is comparable to the process of building biological taxonomies [Lane, 1990]. Specifically, biologists survey existing samples of organisms within the area of study, and then develop theories in order to explain the features that they find – a similar process can be used to develop design spaces in a given domain by identifying features in existing applications in the domain.

There are two important requirements that need to be met for design space analysis to be successful [Geyer, 2000]: the design space analyst must have sufficient knowledge of the

application domain, and the scope of the domain needs to be well-specified. Little guidance is given to exactly how to create a design space using design space analysis. However, [MacLean et al., 1991] provide two sets of heuristics that can guide the process, which are presented in Table 2.3.

Table 2.3: Local and global heuristics for design space analysis.

Local Heuristics	Global Heuristics
Use questions to generate options	Identify options that generate dependencies
Use options to generate questions	Look for novel combinations of options
Consider distinctive or extreme solutions	Design to a set of criteria
Represent positive and negative criteria	Search for generic questions
Overcome negative, but maintain positive criteria	

2.9.5 Design Space Evaluation

Once we have created our design space, we will need methods to evaluate its effectiveness, and hence properties against which to evaluate it. The best classification that we have to break down our design space evaluation discussion is the three types of design space identified by [Gooch, 2013]: generative, evaluative, and conceptual. [Gooch, 2013] suggests a mechanism for evaluating each of these types of design space:

- **Generative**: Generate designs and assess the quality of those designs.
- **Evaluative**: Compare the evaluations produced by the evaluative space with the results of some other value judgement made about the designs.
- **Conceptual**: 'Place' designs in the design space. [Mehandjiev and De Angeli, 2012] used this evaluation mechanism for their design space, where experts in the field were tasked with identifying where their 'favourite' mashup application fit within the space.

Each of these evaluation mechanisms is based on the purpose for which the type of design space is used. For both the generative and evaluative design spaces which are reliant on some judgement of the quality of the outcome of the evaluation, but [Gooch, 2013] makes no suggestion for how this assessment of quality should be made once the design space has been used to generate this outcome.

Other authors have tried to 'place' designs in their design space in order to demonstrate the validity and usefulness of their design space (e.g. [Aghaee and Pautasso, 2013, Minhas et al., 2012, Na et al., 2010, Grammel and Storey, 2010, Brønsted et al., 2010]). [Gooch, 2013] suggests that this is approach is sufficient in demonstrating that the design space is valid, although the lack of any quality judgement means that it is difficult to come to any further conclusions about how 'good' the design space is.

2.9.6 Limitations of Design Spaces

The main limitation of design spaces is that they rely on information already contained within the domain, and thus can be limited in how they can be used generatively [Lane, 1990]. Whilst this may be the case for novelty in the domain as a whole, design spaces can provide designers with insight into areas of the domain that they might not otherwise have considered explicitly.

Design spaces can be vast – containing hundreds of dimensions – and restricting the design space to concentrate on the dimensions that are 'interesting', 'useful', 'relevant', or in some way 'significant' is difficult [Lane, 1990]. Notably, dimensions cannot usually be identified as being redundant until after the design product has been implemented and the design space has already served its purpose.

One of the main limitations of the theory that underpins the theory of design spaces is the view of design as a scientific process, which is argued against by a number of design theorists (e.g. [Schön, 1983, Nelson and Stolterman, 2003]). However, if we consider design spaces in terms of software engineering and software modelling rather than part of traditional design theory, their usefulness becomes more clear.

2.9.7 Design Spaces in the Software Engineering Process

In this section, we discuss the early stages of the software engineering process, across which we believe design spaces can be used. [Geyer, 2000] identifies a link between requirements gathering and design spaces, and clearly the use of the term 'design' in their name has implications as to where in the software engineering process they can be used.

[Sommerville, 2011] identify four stages that all software processes must include:

- **Software specification**: Specifying the functionality and operation constraints of the software.
- **Software design and implementation**: Producing the piece of software to meet the specification.
- **Software validation**: Validating that the software meets the specification, and meets the user/customer's needs.
- **Software evolution**: Modifying the software to meet changes in the user/customer's needs

Design spaces as we have defined them above can be used in any of the first three: [Geyer, 2000] describes the utility of using design spaces alongside requirements, and [Gooch, 2013] identifies that they can be used generatively (in design), and evaluatively (in validation).

A further stage is listed that exists in both the specification and design and implementation stages: *"System modelling is the process of developing abstract models of a system, with*

each model presenting a different view or perspective of a system" [Sommerville, 2011]. Models of existing systems can be used in requirements engineering to motivate requirements for new systems, and models of the new system can be used in design to convey proposed requirements to other stakeholders. These models are normally presented using the Unified Modelling Language (UML). Rather than being an alternative representation of a system, these models should be thought of as an abstraction that conveys ideas from within the design without focusing on every detail [Sommerville, 2011]. Our definition of design space clearly fits this idea of being an abstraction of a software artefact.

Models can be used as an output of the design process (as Model Driven Engineering, or Model Driven Architecture). Advantages of this approach are that the models are at a high level of abstraction meaning that errors are reduced, and the model that is created is reusable and platform independent. The main disadvantage is that the model that is created might not be at an appropriate level of abstraction to be transformed into an implementation. However, such models need not be used in this way: *"So you may create informal design models but then go to implement the system using an off-the-shelf, configurable package."* [Sommerville, 2011]

Two specific types of model highlighted by [Sommerville, 2011] are architecture models and design models. Architecture models define the organisation of the system (e.g. a client-server model), and provide a link between requirements and design. Design models are used to represent the objects or classes that make up a system, as well as displaying the relations between these objects.

We believe that design spaces can be considered as being similar to these models, although design spaces describe the architecture and design rather than presenting this information diagrammatically. Design spaces also provide a single point where these design choices can be described.

2.10 Design

In the last section, we discussed where we believe design spaces can be used in the software engineering process. This section relates the software engineering perspective of design with a more theoretical view of design. We also discuss decision-based design, which is a type of design process that is used more frequently in engineering. Finally we discuss creativity, novelty and design rationale.

No single definition of design is accepted universally [Atwood et al., 2002], although the majority of definitions for design identify it as designers undertaking a process and generating an output [Warr, 2007].

The design process has been defined as a process of making, creating, or inventing [Alexander, 1964, Coyne, 1995, Fallman, 2003, Rasmussen et al., 1994, Warr, 2007]; a conversation or

argument (generally self-reflective) [Rittel, 1984, Schön, 1983, Warr, 2007]; or a problem solving or decision making activity [Thurston, 2001]. The output of the process of design can either by physical or intangible [Warr, 2007], and can result in the creation of something new [Coyne, 1995, Fallman, 2003], or the modification of something that already exists [Jones, 1970, Simon, 1996].

It has been suggested that design can be split into two stages [Beckman, 2007]: analytic (theoretical), where the aim is to gather knowledge [Owen, 1997] and discover existing designs; and synthetic (practical), where the focus moves on to invention or creation.

One theory design for solving design problems is termed 'conservative design', which contains the following stages [Jones, 1970, Warr, 2007]:

1. **Analysing the problem**: *"breaking the problem into pieces"* [Jones, 1970].
2. **Synthesising a solution to the problem**: *"putting the pieces together in a new way"* [Jones, 1970].
3. **Evaluating the outcome**: *"testing to discover the consequences of putting the new arrangement into practice"* [Jones, 1970]

Conservative design views the design process as a structured process that would normally be seen within more scientific disciplines than creative [Fallman, 2003]. Other models for design have been suggested, but these also follow the steps above [Rosson and Kellogg, 1987, Carroll et al., 1979].

Inspiration from other designs also forms part of the design process, as evidenced by [Schön, 1983]'s suggestion that the design process can be described as *"seeing-drawing-seeing"*. The intuition behind this naming is that a designer 'sees' what has been created by other designers in that domain, followed by 'drawing' from the ideas in these designs to create new material, and they can then 'see' their own work and 'draw' on it in an incremental manner.

[Goldschmidt, 1991] also discusses design in terms of 'seeing': *"seeing that"* where new designs are created based upon current knowledge, and *"seeing as"* where designers view and interpret previous designs in the domain in order to generate new designs and new knowledge. Design spaces can facilitate this behaviour by allowing users of design spaces to see what has been done in the domain in order to build upon it.

Engineering literature sees design from a slightly different point of view, where the earliest stage of the design process is known as the *conceptual design phase*, where the general ideas/concepts/features of the artefact to be designed are identified and decided upon [Wood and Agogino, 2005]. Obviously, the decisions that are made at this stage can have a massive impact on the rest of the process, particularly if the decisions made in this conceptual stage turn out to be wrong. In the next section we will discuss a related area of design that is used within engineering – decision-based design.

2.10.1 Decision-based Design

Decision-based Design (DBD) is a type of design process used commonly in engineering that relies on the idea that the design process is based on making a series of decisions [Wood and Agogino, 2005]. It was first suggested by [Shupe et al., 1988], and was later formalised by [Hazelrigg, 1998]. [Hazelrigg, 1998] suggested that design in engineering is made up of two parts: identifying options for solving a problem, and selecting the best of these options. The main issue with DBD is the number of options that are presented within a particular domain, and in particular, the difficulty of searching through all of these options [Olewnik, 2005], a feature which may also relate to design spaces.

One solution to the problem with searching in DBD is to provide a mechanism for ranking the potential choices that can be made [Olewnik, 2005]. Various alternatives exist for this ranking mechanism, used within engineering: Quality Function Deployment (QFD), Pugh's Selection Method, Taguchi Loss Function and Suh's Axiomatic Design [Olewnik, 2005]. Each of these methods seeks to help the designer find the "best" solution for the problem at hand, however none of the methods agree as to what the "best" solution actually is.

Decision-based Conceptual Design (DBCD) is a specialised version of DBD where the decision-making portion of the process happens early on in the conceptual stage, the point at which the process is most uncertain [Wood and Agogino, 2005]. There are three components to DBCD [Wood and Agogino, 2005]:

1. A model of the design space for the entity being designed.
2. Methods for finding the "best" decisions within the design
3. Methods for refining the design space.

If the DBCD process is not sufficiently structured, it may be difficult for the designer to prune the design decisions to the ones that need to be made [Gries, 2004], meaning that greater structure is required.

Decision-based design and decision-based conceptual design have broad similarities to design spaces, since design spaces present the design decisions that a designer might want to make, and hence can support the processes of decision-based design and decision-based conceptual design.

2.11 Creativity & Novelty

Our discussion of design and decision-based design and design spaces is also related to creativity and novelty. This is important to consider because design spaces that are based on the domain as it already exists, meaning that by definition design solutions chosen from a design space cannot be novel in the domain.

Discussions of creativity are normally broken down into three stages [Amabile, 1983]: the creative *process*, the creative *person*, and the creative *product*. [Warr, 2007] defines creativity in design as :

> *"The generation of ideas, which are a combination of two or more existing bundles of knowledge to produce a new knowledge structure. For this idea to be considered creative it should be: novel - unusual or new to the mind in which it arose; and appropriate - conform to the characteristics of a desired or accepted solution. Such creative ideas may then be implemented and embodied in a creative product."* – [Warr, 2007]

There are two different views of the creative process: (a) it is an unconscious process, where the creative person will generate creative ideas when they are not consciously trying to solve them; and (b) it is a conscious process where the creative person actively generates solutions to the problem they are trying to solve [Warr, 2007].

Both views of the creative process rely on an initial phase where the creative person formulates the problem that they are trying to solve, which has been referred to as preparation [Wallas, 1926], or fact-finding [Osborne, 1953]. Once ideas have been generated to solve the problem, both views also suggest an evaluation phase where the creative person assesses the ideas that have been generated in order to determine whether they adequately solve the problem [Wallas, 1926, Osborne, 1953].

The idea generation phase is where the views of the conscious and unconscious views of creativity differ. In the unconscious view, the problem is *incubated* by the creative person's unconscious mechanisms, followed by an *illumination* phase where ideas are generated [Wallas, 1926]. In the more conscious model, the problem is processed by the creative person and new ideas are generated by combining combinations of existing ideas [Osborne, 1953].

Other views of the creative process rely on some understanding of the creative person who is carrying out the creative process. In particular, the three aspects that need to be understood are: domain skills/knowledge, creative skills/knowledge, and task motivation/enthusiasm [Amabile, 1983]. There are five stages to this model of the creative process:

1. **Problem and task presentation:** The creative person receives the problem and tries to understand what is involved. High task motivation is required in order to continue the process, which may be motivated by domain skills or knowledge) [Amabile, 1983, Warr, 2007].
2. **Preparation:** The creative person seeks to acquire enough knowledge about the problem and potential solutions. High domain skills are important in the preparation phase [Amabile, 1983, Warr, 2007].
3. **Response generation:** Potential solutions to the problem are generated. Creative skills and task motivation are both important at this stage as the frequency of ideas generated is likely to be higher given these skills [Osborne, 1953, Amabile, 1983, Warr,

2007]

4. **Response validation:** Validation and evaluation of the responses that are generated by the creative person, which benefits greatly from prior domain knowledge so that the creative person is accurate when validating or evaluating their ideas [Amabile, 1983, Warr, 2007].
5. **Outcome:** The creative product is generated.

Other authors have also ascribed great importance to the knowledge of the creative person who is undertaking the creative process, since they are equipped to combine the knowledge that they have in new ways [Csikszentmihalyi, 1997, Warr, 2007]. On the other hand, novice designers require some additional knowledge [Christiaans and Venselaar, 2005].

By definition, the outcome of the creative process is the creative product. Evaluating how creative the creative product is has been suggested as being an important measure for creativity [Amabile, 1983, Gilchrist, 1972]. One aspect of this measurement is the novelty of that creative product, which is defined as the creative product's 'unusualness' [Gilchrist, 1972].

Novelty has been suggested as being relative: a concept can be novel relative to the whole domain, or novel relative to an individual. These concepts are referred to as historical novelty (h-novel) and psychological novelty (p-novel, respectively [Boden, 1996]:

- **H-novel:** Historically (h-)novel ideas are those that are entirely new to the domain.
- **P-novel:** Psychological (p-)novel ideas are those that are already known or used in the domain, but are novel to the creative person.

Analogous concepts exist directly in creativity: historical (h-)creativity and psychological (p-)creativity. Definitions of these terms follow from the definitions of h- and p-novelty, where psychological creativity is ascribed an entity that is seen as creative to the creative person, whereas historical creativity is ascribed to an entity that is seen as creative to society as a whole [Warr, 2007]. P- and h-creativity are referred to using a number of different terms, respectively: "small 'c' creativity" and "big 'c' creativity" [Csikszentmihalyi, 1997], or "impromptu or personal creativity" and "revolutionary creativity" [Shneiderman, 2000]. Shneiderman also suggests an intermediary form of creativity, termed "evolutionary creativity", which is a little-and-often approach to creativity, that may lead to revolutionary creativity over time [Shneiderman, 2000].

Creativity and novelty are both important parts of the design process, and we need to ensure that our work does not inhibit them. We have seen that both the design and creative processes rely on a stage in which the designer aims to become familiar with the domain in which the design artefact is contained.

It is clear that design spaces – as any artefact that presents a list of problems and solutions – are unlikely to prompt designers to solve other problems that are not already within that list. However, we believe that with the right support, we can help to support the designer with

h-novel ideas, as well as using the design space to provide them with p-novel ideas that they might not have considered otherwise. The effects of design spaces and novelty are discussed in Chapter 6.

2.12 Design Rationale/Criteria

Our final discussion of design focuses on design rationale. Design rationale are defined as explanations of the reason for the design of a particular part of an artefact [Gruber and Russell, 1991]. Rationale normally include the reasons the design was chosen over other potential designs, and other information such as expected behaviour [Gruber and Russell, 1991]. It can be summarised as follows: *"In short, a design rationale explains the 'why' of a design by describing what the artefact is, what it is supposed to do, and how it got to be designed in that way"* [Gruber and Russell, 1991].

Design rationale aim to describe the following topics [Gruber and Russell, 1991]:

1. What is the purpose of the entity being justified, and how does it work?
2. What is the expected behaviour of the entity being justified?
3. Why was the entity designed this way? What were the requirements for the entity and the assumptions made about it?
4. What other alternative designs were considered for the entity?

Whilst it is helpful to answer all of the questions above, it is not always possible for the designer to be able to keep track of these at every stage in the design process, particularly if the knowledge of these areas is implicit [Gruber and Russell, 1991]. Recording these rationale is made more difficult because of the range in the abstraction level of design decisions being made – high-level and low-level design decisions are often made at the same time [Guindon, 1988]. Furthermore, designers often require assistance in recording design rationale [Gruber and Russell, 1991].

Design rationale are important to us because they have been supported as part of [MacLean et al., 1991]'s design space concept: Questions, Options, Criteria (QOC) . Design rationale are presented as criteria, which have the following set of properties:

- A criterion measures a property of the artefact being designed that the designer has indirect control over by exercising choices over options.
- A criterion must be *unconditional* in the sense that, other things being equal, the greater the extent to which the criterion is met, the better the design.
- A criterion must be *evaluative*; that is, it must be a measure of some property of the artefact, with a definite sense of higher assessment values being better. (It can be convenient to think of a criterion as potentially yielding a quantitative value, even if only on an ordinal scale.)

Providing criteria in the design space means that the design artefact that emerges from the design space contains all of the rationale for the design decisions chosen for that design, as well as reflecting the through processes of the designer that created the design [MacLean et al., 1991]. Recording these rationale is particularly important when large systems are being designed and a number of designers might be making design decisions, since designers need to gain some insight as to why other design choices might have been made [MacLean et al., 1991].

We believe that the inclusion of criteria within the representation of the design space can also limit the expressiveness of the design space, in that criteria being applied at an early stage in the process may restrict what the designer may choose if they are using the design space in a generative way. Hence criteria/rationale may be more useful to be part of the tool that supports the user of the design space rather than being a component of the design space itself.

2.13 Design Spaces and End-user Service Composition

Our research goal of creating a design space for EUSC applications was motivated initially by a number of design spaces that were created in SC, EUSC, or the closely related domain of mashups. In this section, we will provide an overview of each of these design spaces.

We identified 11 different design spaces in the SC domain:

1. Reusable Decision Space [Aghaee et al., 2012]
2. A Survey of Mashup Development Environments [Grammel and Storey, 2010]
3. Mashup Evaluation Framework [Minhas et al., 2012]
4. Study of Mashups as a Software Development Technique [Na et al., 2010]
5. Analytical Framework [Mehandjiev and De Angeli, 2012]
6. Mashlight [Albinola et al., 2009]
7. Mashup Framework Categorisation Model [Fischer et al., 2009]
8. Application Composition at the Presentation Layer [Pietschmann et al., 2010]
9. Service Composition and Pervasive Computing [Brønsted et al., 2010]
10. Building mashups by demonstration [Tuchinda et al., 2011]
11. Picozzi's Mashup tool classification [Picozzi, 2014]

For each of the different SC/EUSC design spaces, we consider the following questions:

Q1. Was the design space an explicit aim of the work, or a side effect?
Q2. What did they model with the design space?
  Q2.1. What are the elements of the design space?
  Q2.2. How many dimensions does the design space have?
  Q2.3. What do they call their design space?

Q3. How did they create the design space? Does it follow any of the modelling techniques we have already identified?
Q4. What did they use the design space for?

We identified these questions as being important to answer for the existing SC/EUSC design spaces because they provide us with an important reference for the structure, contents and creation of our design space.

2.13.1 Reusable Decision Space [Aghaee et al., 2012]

[Aghaee et al., 2012] present a "reusable decision space", the creation of which was an explicit aim of the work. Specifically, they sought to highlight the variability in the design of MDEs.

Their design space is based on a meta-model outlined by [Nowak and Pautasso, 2011]. The meta-model is presented as two layers: an abstract layer that describes artefacts and relationships that can exist between artefacts, and a concrete layer that is specifically focussed on the design space itself. The concrete layer specifies that artefacts are either design issues or alternatives, and the relationships are "solved by", "influences" and "depends on" [Nowak and Pautasso, 2011]. The model is shown in Figure 2-11.

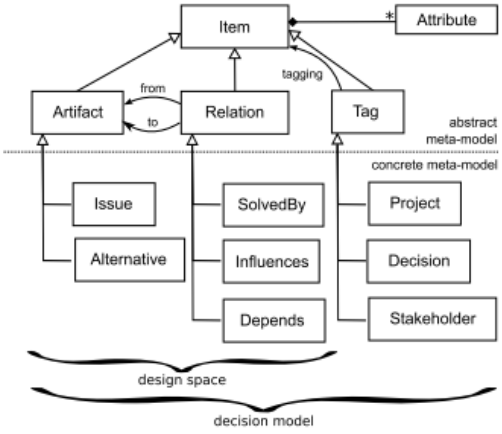


Figure 2-11: [Nowak and Pautasso, 2011]'s meta-model

[Aghaee et al., 2012]'s decision space contains 9 dimensions and 27 potential solutions across these dimensions. The dimensions themselves are separated into three groups: issues relating the community, environment-specific issues, and language-specific issues. The use of the term "decision space" is clearly related to design space, since the name references the decisions that the designers of the Mashup Development Environments (MDEs) have made, or could have made.

The decision space was created by surveying a collection of existing MDEs. The surveyed applications were used where they were available, and if not then they were read about in literature or other available sources if no literature was available [Aghaee et al., 2012]. The issues found in these applications were then collated, overlapping issues were removed and the set as a whole was organised into distinct groups. For each design issue, they identify the benefits of each solution, challenges associated with it, as well as an example of that solution being used [Aghaee et al., 2012]. As well as being discussed in detail, the relationships between the various decisions and potential solutions are presented graphically.

The final part of the paper presents the results of using the decision space to profile existing MDEs. They identify a candidate list of 60 MDEs, which is reduced to 22 based on the availability of resources to accurately perform profiling on these MDEs. The results of this profiling activity are presented as a feature matrix that is presented in Figure 2-12. The profiling activity was seen to be a validation mechanism for the contents of the design space, in that they were able to verify that all potential solutions within the design space were used within at least one of the profiled MDEs [Aghaee et al., 2012].

2.13.2 A Survey of Mashup Development Environments [Grammel and Storey, 2010]

[Grammel and Storey, 2010] surveyed available MDEs to assess the support for EUD in MDES, and to discuss how mashups might be able to help support the "Smart Internet". As such, the design space is not an explicit output of the work, but instead facilitates the discussion of MDEs, EUD, and the Smart Internet.

They describe their method for creating their design space as a "qualitative, exploratory tool analysis". This process involved identifying a series of appropriate MDEs, and evaluating each of those applications for a period of 3-6 hours [Grammel and Storey, 2010]. Evaluations included creating mashups with the application, exploring its different sections or features, and reading related tutorials or associated documentation. The process was limited by the fact that the evaluator only recorded the topics or features that were relevant to EUD themes.

The design space itself is based on 6 different themes that are normally associated with EUD- and mashup-based problems, where each of these themes had a series of questions as a guide for the evaluation. The 6 themes that make up the main topics of the design space are: levels of abstraction, learning support, community features, discoverability, UI construction, and software engineering techniques [Grammel and Storey, 2010]. As well as these themes, the 'type' of mashing-up that is supported was also assessed, i.e. whether the application supports process, data or Website mashing-up.

[Grammel and Storey, 2010]. present their design space firstly as a discussion of its contents, describing each category, along with the decisions and potential solutions that are contained within it, along with a list of the profiled MDEs that implement that solution. The same data

Name	Piggy Bank [27] (2005)	FeedRinse (2006)	Dapper (2006)	JackBe Presto (2006)	Swashup [36] (2007)	d.mix [23] (2007)	JOpera [45] (2007)	Yahoo! Pipes (2007)	Karma (2008)	SABRE [35] (2008)	Kapow Katalyst (2009)	MashArt [12] (2009)	Lively Wiki [33] (2009)	RoofTop [25] (2009)	IBM Mashup Center (2009)	ServFace Builder [42] (2010)	VikiBuilder [24] (2010)	Microsoft Visio [52] (2010)	Husky (2011)	DashMash [9] (2011)	Petals BPM (2011)	Sqwelch [20] (2011)
Specificity																						
Generic	+	+	+		+	+	+	+	+									+	+	+		
Specialized				+						+	+	+	+	+	+	+	+				+	+
Target End-User																						
Local Developers				+			+				+											
Non-Programmers	+	+	+			+		+	+	+		+	+	+	+	+	+	+	+	+	+	+
Programmers					+																	
Automation Degree																						
Full Automation	+																					
Semi Automation		+	+	+	+	+	+	+	+	+	+	+	+	+	+	+	+	+	+	+	+	+
Liveness																						
Level 1																		+				
Level 2																					+	
Level 3	+	+	+	+	+	+	+	+	+	+	+	+			+	+	+		+			+
Level 4													+	+						+		
Online Community																						
Private				+										+	+							
Public		+						+			+											
Collaboration																						
Blackboard																						+
Fork and Edit				+				+							+							
Wiki													+									
Interaction Technique																						
Editable Example						+																
Form-based		+		+			+	+			+	+			+	+	+					+
PbD			+						+													
Spreadsheets									+										+			
Textual DSL				+	+																	
Visual Language				+			+	+		+	+	+		+	+		+			+	+	
WYSIWYG													+	+	+	+		+				+
Visual Language																						
Iconic										+							+					
Wiring				+			+	+			+	+		+	+					+	+	
Control Flow																						
Explicit							+				+										+	
Implicit				+				+				+		+	+					+		

Figure 2-12: Application profiling results using Aghaee's decision space. (from [Aghaee et al., 2012])

are then presented as a feature matrix, similar to that presented by Aghaee et al. [Aghaee et al., 2012, Grammel and Storey, 2010]. Following the presentation of the results of profiling, they discuss deficiencies in some of the profiled applications, as well as presenting potential areas of future work within any of the topics that make up the design space.

2.13.3 Mashup Evaluation Framework [Minhas et al., 2012]

[Minhas et al., 2012] created a "Mashup Evaluation Framework" in order to gauge and improve the understanding of MDEs in terms of the concepts involved, characteristics they have, and user expectations. They also created a design space to compare and analyse how currently available MDEs operate. The aim of their work is to evaluate the current state-of-the-art in mashup development applications and then identify potential areas for future research that fall within mashup design or EUD.

The method that [Grammel and Storey, 2010] use to create their evaluation framework is not described in any detail. It is stated as being "qualitative and exploratory", and is presented in two stages. The first stage focuses on high-level concepts such as the type of mashup development application, and the target user of the application (enterprise or consumer). The second stage is much more in depth and is presented across the three dimensions stated above.

The framework presents different concepts within the mashup domain, which are split into three categories: design features, usability, and technical profile [Minhas et al., 2012]. Each category is either made up of a series of decisions with corresponding solutions as potential solutions, or simply a series of solutions to solve the problem associated with the higher-level category. For instance, consider mashup design features vs. usability: mashup design features contains two design decisions – mashup activities (3 solutions), and mashup technique (5 mutually exclusive solutions); usability is made up of 6 solutions with no intermediary decisions.

They evaluate the design space by profiling 12 MDEs that were available at the time, either available in production or described in mashup/SC literature [Minhas et al., 2012]. The results of the profiling exercise are presented as a feature matrix (as [Grammel and Storey, 2010, Aghaee et al., 2012]). After presenting the results of application profiling, they go on to discuss each of the areas of their framework and identify areas in which knowledge is lacking, and those areas that are applicable to EUD. The main finding of their work is that a more user-oriented design is required, particularly when these applications are meant to be code-free [Minhas et al., 2012].

2.13.4 Study of Mashups as a Software Development Technique [Na et al., 2010]

[Na et al., 2010] carried out a study to assess the support provided by MDEs in various approaches from EUD. Two small design spaces were created as an intermediary stage in this process, although only one of which is evaluated.

As with the design space created by [Minhas et al., 2012], it is not clear how the features were identified, or how the design space as a whole was created. [Na et al., 2010] state that "both qualitative and quantitative research methods have been utilized".

The first design space they create is the smaller of the two, only containing three different dimensions: architecture, combined items, and involved users. This design space is never evaluated. The second design space presents features of MDEs across a number of different dimensions, some of which are well-defined, others are not. 'Programming skill' is an example of a concrete, well-defined dimension, whereas 'Service' is not well-defined, as there is no consistency in the solutions that can be chosen to solve this dimension.

The second, larger design space was evaluated by profiling 8 different MDEs. The selection process of the applications used statistics describing API usage from Web-based sources such as ProgrammableWeb[14] in order to identify which MDEs should be chosen for profiling [Na et al., 2010]. 8 MDEs were chosen for profiling, and the results of this are presented in the form of a feature matrix. After the presentation of the feature matrix, the analysis focuses on how support for EUD is achieved in MDEs, and where this support is lacking and could be improved.

2.13.5 Analytical Framework [Mehandjiev and De Angeli, 2012]

[Mehandjiev and De Angeli, 2012] create an analytical framework with the aim of assessing the support for EUD techniques in MDEs. In particular, they sought to analyse how MDEs can be designed to better support non-programmers – particularly those with low technical skills – and to highlight the areas of mashup design that require further research [Mehandjiev and De Angeli, 2012].

The design space has two axes that are identified from prior research into EUD and Service-oriented development [Mehandjiev and De Angeli, 2012]. The Service-oriented axis is made up of a collection of aspects related to the life cycle of service composition, derived from prior literature [Papazoglou and Georgakopoulos, 2003, Ramollari et al., 2007, Mittal, 2010]. The EUD axis contains aspects that were identified from prior work on the design activities that are performed by end-user designers with little technical knowledge (e.g. the creator of a spreadsheet in MS Excel or similar). Also considered were meta-design activities –

[14] `http://www.programmableweb.com`

the activities of those who create the application being used by the non-technical end-user designer (e.g. the creators of MS Excel or similar).

Their framework was evaluated by experts at a session in a workshop discussing EUD and service-oriented systems. At the session, attendees were asked to apply the framework to their 'favourite' mashup application. This was achieved by asking participants to list features of their favourite MDE, or of MDEs with which they were familiar. They were asked to 'place' these features within the framework, as well as discussing when they encountered difficulties in placing the features within the framework [Mehandjiev and De Angeli, 2012].

2.13.6 Mashlight [Albinola et al., 2009]

[Albinola et al., 2009] present the creation of a MDE called Mashlight. During the process of designing and creating Mashlight, they perform a review of industrial MDEs in order to classify design choices identified within them. They state that their approach should result in a mashup application that is usable by anyone, regardless of technical expertise, to create any of the three different types of mashup (process mashups, data mashups or Web/UI mashups), without any reduction in what is possible with the application [Albinola et al., 2009].

It is not explicitly stated how their model was created, but it is clear that they were motivated by a requirements gathering exercise, since the requirements stated later in the work are based on similar topics to the dimensions of the design space [Albinola et al., 2009].

[Albinola et al., 2009]'s design space is made up of two dimensions: mashup type and target user. Each dimension has only three possible values, meaning that the design space is very limited in scope.

The design space is never explicitly evaluated, although its use as a motivation for requirements for the mashup development application that is described in the paper: Mashlight. The design space is also used as a profiler by classifying 17 examples of existing industrial mashup development applications and associated Web technologies [Albinola et al., 2009].

2.13.7 Mashup Framework Categorisation Model [Fischer et al., 2009]

[Fischer et al., 2009] produce a design space as an overview of the main characteristics of a set of MDEs in response to recognising a rise in the popularity of MDEs for users with little to no technical skill.

They do not explicitly state how they gathered the concepts identified within their small design space, but the analysis is claimed to be a representative overview of the mashup space as a whole, both in industrially available mashup development applications and current research [Fischer et al., 2009].

[Fischer et al., 2009]'s design space is made up of two axes: one based on the technique that the MDE employs to create mashups (also linked with the level of automation provided), and the other based on attributes of the target user. The 'user' dimension has three potential values: developer, power user and casual user. The mashup technique dimension is categorised by automation level: automatic creation, semi-automatic creation, and manual creation.

The design space was not evaluated in its own right, but the results of profiling 32 industrial MDEs were used in order to identify the limitations of current mashup-creation approaches, which can then identify areas that require more research [Fischer et al., 2009]. Their future work is said to be to use the results of the classification process to create a MDE.

2.13.8 Application Composition at the Presentation Layer [Pietschmann et al., 2010]

[Pietschmann et al., 2010] create a design space with the aim of investigating potential solutions to the problem of SC at the presentation layer. Their 'solution space' was created to help identify challenges and problems that are yet to be solved in the domain. The aim of the design space is to systematically compare the different approaches to SC at the presentation layer. The methodology used to create the design space is not discussed.

The design space is split into three categories: the component model, the composition model, and the design/execution environment. Each of these categories is made up of a number of dimensions:

- Component model: component types, data format, description.
- Composition model: internal component model, composition logic, layout logic, output types.
- Design/execution environment: target user, design paradigm, deployment/execution, additional features.

Their design space is evaluated by profiling three different MDEs in order to identify areas that require further research or development [Pietschmann et al., 2010].

2.13.9 Service Composition and Pervasive Computing [Brønsted et al., 2010]

In work focused on SC for pervasive services, [Brønsted et al., 2010] create a design space based on "variation points" in the surveyed SC applications and how these variations could be utilised in the composition of pervasive services.

Whilst investigating the goals of designers of pervasive SC applications, [Brønsted et al., 2010] identified a number of variation points: 'points' in the design at which there is variation across different applications. They state that the variation points were identified

when considering the elements involved in the SC process, but it is not clear how these variation points were identified.

The design space is split across three categories: specification, runtime and deployment. Specification detailed aspects such as who it is specified by, when and where it is specified, whether the specification is implicit or explicit, and whether Quality of Service (QoS) is included [Brønsted et al., 2010]. The runtime category contained management of contingencies, and the minimum device required for either the framework or for running a service [Brønsted et al., 2010]. Deployment contained network aspects such as infrastructure, topology, and how the framework was evaluated [Brønsted et al., 2010].

They used the design space to profile 24 available SC and mashup applications from academic work. The results of this profiling exercise were then presented as two feature tables. They also sought to assess the impact of the works that presented the profiled applications by counting the number of citations of each respective piece of work [Brønsted et al., 2010].

2.13.10 Building mashups by demonstration [Tuchinda et al., 2011]

[Tuchinda et al., 2011] aimed to create a MDE that could better allow end-users of any level of technical skill to create mashups. To achieve this, they categorised other MDEs in order to identify features that could be used in the application to be created.

This design space was created based on categorising existing MDEs and the features that they implement. To generate the categorisation, they reviewed the 50 most popular mashups on ProgrammableWeb[15].

[Tuchinda et al., 2011]'s design space is made up of 5 different dimensions, each focussing on a different aspect of mashup technologies. These are data retrieval, source modelling, data cleaning, data integration, and mashup type supported.

The design space is evaluated by profiling 13 existing MDEs used either in industry or in academia. The results of this profiling exercise are presented as a feature matrix. Following profiling existing applications, the results of the categorisation are used to motivate requirements for and the design of their own mashup development application, Karma [Tuchinda et al., 2011].

2.13.11 Mashup Tool Classification [Picozzi, 2014]

[Picozzi, 2014] classified 10 mashup development tools including one tool (PEUDOM) for which they document the creation process. This classification was created to try to identify the 'placement' of PEUDOM with respect to the other classified mashup tools.

[15] `http://www.programmableweb.com`

This design space contains 5 dimensions, relating to properties of mashup development tools that they discuss earlier in their work: mashup types, component type, component description, integration logic, and advanced techniques. The design space is not evaluated.

2.14 Chapter Summary

In this chapter, we introduced a number of topics within End-User Service Composition (EUSC) and design spaces. We also discussed a number of examples of design spaces that have been created within the EUSC domain to motivate the work in this thesis.

Our discussion of EUSC first introduced Service Composition (SC): the underlying process upon which EUSC is based. We described what services can be composed, and distinguished between the services that are coordinated with one another (components) and the output of the composition process (composites). We then identified the distinct stages that make up the process, and discuss who undertakes each of these stages.

We then focused on aspects that relate directly to end-users and EUSC, providing relevant background relating to end-user development (EUD), and mashups. The last part of our EUSC discussion introduced a number of EUSC applications that will be used in several stages in the thesis, before introducing prior approaches to gathering requirements in EUSC.

The second section of the literature review introduces design spaces, and how other researchers have defined and used them previously. We discuss what design spaces can contain, how they can be used, and where in the software engineering process they might be used. Next, we discuss more general topics in design and how design spaces relate to aspects such as creativity and novelty.

The final section of the literature review describes the confluence of these two research areas by identifying how design spaces have been used in EUSC. We describe a number of prior EUSC design spaces in terms of what they contain, how they were created, and how they were evaluated.

Given that EUSC is an ill-structured problem and the range of features identified in available EUSC applications, it is not clear what the best approach is. We identified two studies that sought to gather requirements for EUSC applications, as well as highlighting some deficiencies in these approaches. Thus, we identified that more thorough specification is needed in the domain. This motivates our first research goal:

RG1: Derive and evaluate a set of requirements for an EUSC application.

The creation of a number of design spaces for EUSC highlights that a number of authors independently believe that they are useful tool to be applied to design spaces [Aghaee et al., 2012, Grammel and Storey, 2010, Minhas et al., 2012, Na et al., 2010, Mehandjiev and De Angeli, 2012], etc. We also identified prior work that suggests a number of ways in

which design spaces might be used in the software engineering process [Gooch, 2013]. This motivates our second research goal:

RG2: Create and evaluate a Design Space for EUSC applications.

Design spaces can grow to be very large, and often require tool support to allow designers to interact with them [Baum et al., 2000], in particular creating them, using them to evaluate designs and to generate new designs. This motivates our final research goal:

RG3: Create and evaluate a Design Space Tool to facilitate the design and creation of design spaces and domain applications.

By completing each of these research goals, we will be in a position to address our overall aim: to explore the use of design spaces in design in the software engineering process.

CHAPTER 3
ESTABLISHING REQUIREMENTS FOR EUSC APPLICATIONS

3.1 Chapter Introduction

Requirements engineering is the first stage of the Software Engineering process [Sommerville, 2011]. EUSC is a domain in which there are relatively few instances of prior work that have sought to gather requirements for EUSC applications. In Chapter 2, we discussed EUSC, design spaces, and the potential uses of design spaces within software engineering. We identified that design spaces can be used in the specification stage of software engineering, and that they can be used to help present requirements that are gathered in this specification stage [Geyer, 2000].

Requirements engineering – the derivation of requirements to which a piece of software must adhere – is an important part of the specification stage of the Software Engineering life cycle [Sommerville, 2011], and it is clear from the available SC and EUSC applications we reviewed in Section 2.7 that there are a myriad of possibilities in the design of EUSC applications. There has been a small amount of research into gathering requirements for SC applications specifically [Mehandjiev et al., 2010b, Namoun et al., 2010b], as well as few instances of requirements for such tools even being specified before describing the implementation of an EUSC application or MDE [Cappiello et al., 2011b, Albreshne and Pasquier, 2011, Albinola et al., 2009, Aghaee et al., 2012]. The prior requirements sets for EUSC were described in Section 2.8. We believe that this lack of focus on EUSC requirements shows that further investigation is required into needed for applications within this domain.

This chapter presents a study carried out to address our first research goal: to derive and evaluate a set of requirements from potential end-users of End-user Service Composition applications. We describe our study that is based on an adaptation of the Scenario-based Requirements Analysis Method (SCRAM): a requirements engineering method based on using scenarios and a concept demonstrator in order to elicit requirements from participants.

The resources associated with this chapter are provided in Appendices A-C.

Defining a set of requirements for EUSC is an important first step in understanding what end-users want them to be able to do. We believe it particularly important in this case because it is an area in which there are a very limited set of requirements available in prior work, and it the area would benefit from a large, coherent set of requirements.

Throughout the chapter, we describe the elicitation process using some of the derived requirements as exemplars. The list of all 139 requirements are presented in Appendix C. A version of the work described in this chapter was first published in [Ridge and O'Neill, 2014].

The overall aim of this chapter is to derive and evaluate a comprehensive set of requirements for an EUSC application. Since the focus of our work in this thesis is on EUSC rather than SC, we aim to gather our requirements from potential end-users of EUSC applications rather than seeking out highly technical or business customers who could provide requirements for SC in general. Both SC and EUSC implementations that are described in prior literature tend to follow a technology-driven specification (named Web Service tunnel vision [Edmond et al., 2005]), and we feel that such specifications would benefit from a user-driven approach to definitively capture what users want rather than what the traditional service-oriented technologies can provide. Typical requirements gathering approaches focus on business customers rather than end-users as consumers [Sommerville, 2011], and as such existing methods require some adaptation before being suitable for our task.

This aim can be broken down into two concrete objectives:

1. Create and carry out a generalisable, repeatable method for gathering requirements that is more suitable for our domain.
2. Derive a coherent and robust set of requirements for an EUSC application.

3.2 Requirements Gathering Techniques

Typical user-focused requirements gathering techniques tend to focus on building a product for a business customer rather than being targeted at consumers as end-users and as such are not suitable to be used out-of-the-box in the EUSC domain [Sommerville, 2011]. Our goal is to gather requirements from a group of potential end-users of an EUSC tool, so if we decided to use any of these established techniques, they are likely to require some modification to be suitable for use in EUSC.

There were a number of requirements gathering approaches that we identified as being suitable base for the method we would use to gather requirements. Two of the suggested methods are classified as information gathering methods [Maguire and Bevan, 2002], which is the first stage of requirements gathering, where requirements engineers seek to find

information about users, other stakeholders and the processes that need to be supported by the application. The suggested methods were diary keeping and field studies.

Diary keeping provides a mechanism for users to keep track of their own behaviour over a period of time [Maguire and Bevan, 2002]. The Requirements engineers are then able to evaluate users' tasks in order to identify requirements. The main disadvantage of diary studies is that participants often forget to record activities in the diary [Maguire and Bevan, 2002].

Field studies are an alternative approach to identifying users' current tasks, but with less effort than is required for the user when they are recording a diary [Rieman, 1993]. Observations can either be carried out directly by a requirements engineer, or by recording video which can be analysed later [Maguire and Bevan, 2002].

Since our participants were unlikely to already be familiar with EUSC, information gathering needed to operate in both directions: as well as requirements engineers gathering information from participants, our participants need to be able to gather enough information from the requirements engineers to be able to understand the domain.

Once the requirements engineers have gathered information that they require about the domain, they need to analyse the needs of the potential users of the system to be designed [Maguire and Bevan, 2002]. Potential needs identification methods are surveys, interviews, and focus groups.

Interviewing is a process that is often used when gathering requirements from potential end-users [Sommerville, 2011]. Interviews in requirements gathering would normally involve a stakeholder who is already using a similar type of system, and hence they would have some knowledge as to what answers are "right" or "wrong" in discussions with the requirements engineer [Sommerville, 2011]. Furthermore, these interviews would normally be an opportunity for the requirements engineer to obtain information about the domain in which the requirements are being gathered [Sommerville, 2011], whereas in this study is the requirements engineer who is knowledgeable in the domain, and the interviewee who may not be.

Surveys can be thought of as an alternative form of interview, where users or stakeholders provide written answers to a set of questions rather than being in a verbal conversation with the requirements engineer [Maguire and Bevan, 2002]. Surveys can be particularly useful in gathering both quantitative and qualitative data about the system to be developed [Maguire and Bevan, 2002].

Focus groups are similar to interviews, except carried out in a group setting [Rabiee, 2007]. Focus groups can be more useful than interviews in that they provide a range of feelings that participants might have about the topics being discussed [Rabiee, 2007]. However, they may fall foul of existing relationships between participants affecting their willingness to participate in discussions [Rabiee, 2007].

None of the requirements gathering methods listed above meet our need of providing participants with enough information to be able to understand EUSC enough to be able to suggest requirements for an EUSC application. To facilitate this passing of knowledge to participants, we also considered the resources that can have been used to familiarise participants with a domain.

3.2.1 Scenarios and Use cases

Scenarios and use cases can be used alongside other user-focused requirements gathering mechanisms to help identify the tasks that the prospective piece of software will need to support [Sommerville, 2011]. They help to identify how the user might carry out the required tasks that the prospective application supports and help the user better understand it. Scenarios and use cases are both sequences of events that follow some task-based narrative, although scenarios provide more context and motivation to help the user visualise the process.

In our case, the aim of scenarios would also be to ensure that participants in our requirements gathering study have an adequate understanding of EUSC and what an EUSC tool does, since it is a relatively young software area with which few people are familiar. We identified a single requirements gathering method that used scenarios to familiarise participants with a domain: the Scenario-based Requirements Analysis Method.

3.2.2 The Scenario-based Requirements Analysis Method (SCRAM)

The Scenario-based Requirements Analysis Method (SCRAM) is a requirements analysis method that uses aids including scenarios, a concept demonstrator application and a number of potential designs to introduce participants to the domain, and ultimately gather requirements for a new tool in that domain [Sutcliffe, 2003, Sutcliffe, 1998, Sutcliffe et al., 1998]. The overall process of SCRAM is made up of four stages:

1. **Domain familiarisation and preliminary requirements capture:** In order to design and implement a prototypical demonstrator application, the requirements engineer needs to be familiar with the domain, and gather an initial base set of requirements to which this demonstrator application must adhere.
2. **Storyboarding and design:** The prototypical demonstrator to be used in the study sessions is designed and created alongside scripts that will be used in study sessions to show participants each of the major features of the demonstrator. Other designs are also provided to outline the process that might be undertaken if the demonstrator were a full-fledged application. Alternative designs can also identify points of possible deviation from the design of the demonstrator.
3. **Requirements exploration (study sessions):** Participants are presented with the concept demonstrator, alongside scenarios, a script, and alternative designs in order

to convey to them how the demonstrator might be used in a 'real world' situation. Participants are given probe questions at key points throughout the script in order to elicit responses, and hence requirements. The design decisions that were made in the demonstrator are illustrated with design rationale documents.

4. **Session analysis**: The data gathered from the SCRAM sessions is then analysed in order to derive requirements that can then be reported back to participants.

We chose SCRAM as the base upon which we could build our method because it is already a robust method that supports requirements gathering and domain familiarisation for the participants of the study. Whilst it is the closest method to what we need for this requirements gathering exercise, it is not clear how well the method generalises to end-users as participants rather than business customers.

In the next section, we discuss the changes that were made to this method to ensure that it better suited our needs, as well as extensions that describe exactly how requirements were elicited, which is not detailed in the original method.

3.3 Methodology

This section discusses our tailored requirements analysis method. It covers the stages of the method before, during, and after study sessions detailing the preparation that was required, session instructions, and analysis of the output of the study sessions, respectively.

The basis of our requirements gathering method is SCRAM (described in Section 3.2.2). The guidance for applying SCRAM only describes the process up to the end of the requirements gathering study sessions, with no suggestions for requirements elicitation from the data that is obtained from participants, other than the fact it should occur at the end of the session [Sutcliffe, 2003, Sutcliffe and Ryan, 1998].

We separate our discussion of the method that was used into three sections:

1. **Pre-study**: The activities that were performed before study sessions, including specification, design and implementation of the prototypical demonstrator application; creation of a demonstration script along with probe questions; and scenario generation.
2. **Study**: The activities that were carried out with participants in the study sessions themselves.
3. **Post-Study**: The activities that were performed in order to transform the data that was obtained from the study sessions into a set of concrete requirements for EUSC tools.

3.3.1 Pre-Study Method

SCRAM requires the creation of a number of additional materials to be used in the sessions themselves, including:

- A *prototypical demonstrator application* that is treated as an interactive script to guide participants through the EUSC process during the sessions. The creation of the demonstrator application must be preceded by a preliminary requirements capture and feature specification.
- A *demonstration script* to specify the actions that should be completed and the probe questions that should be asked.
- *Alternative designs* to the one that was used in the demonstrator to provide participants with other options that could have been chosen.
- One or more *scenarios* to convey the problem area to the participant.

Preliminary Requirements Capture

To create a prototypical demonstrator to be used in SCRAM sessions, it first needed to be specified, designed, and implemented. The first stage in this process is a preliminary requirements gathering phase to provide an initial specification for that prototype.

A preliminary requirements capture is identified by [Sutcliffe, 2003, Sutcliffe and Ryan, 1998] as the first stage of SCRAM, but little insight is provided as to how these requirements might be gathered. What insight that is given assumes that requirements are being gathered in a business context, which is a common approach in discussions of requirements gathering. Clearly this necessitates a different approach in our case.

Our preliminary requirements capture was performed through a literature review. The first stage in this review was to identify other studies that aimed to gather requirements for EUSC [Mehandjiev et al., 2010b, Namoun et al., 2010b]. Due to the small number of requirements found in these approaches (11), we expanded our search to include literature that listed requirements for EUSC applications where they do not specify the process by which they were obtained. Finally, we performed a review of the domain in order to ensure that all aspects of the prototypical demonstrator were specified adequately enough that they could be implemented. Topics that were identified in this preliminary requirements capture ranged from general SC [da Silva et al., 2008], Web-based SC [Nestler et al., 2011], mashup development applications [Aghaee and Pautasso, 2012, Albinola et al., 2009], and pervasive SC [Bottaro et al., 2007]. Our preliminary requirements are listed in Appendix B, and are grouped into functional and non-functional categories.

Demonstrator Specification

The second stage in the pre-study method is to specify and design the prototypical demonstrator that will be presented to participants within the study sessions [Sutcliffe, 2003, Sutcliffe and Ryan, 1998]. This prototype is treated as a script that guides participants through the process in order to gather requirements, and can have limited interactivity or functionality

[Sutcliffe and Ryan, 1998]. However, [Sutcliffe, 1998] indicated that responses from participants tend to be of higher quality if the prototype is interactive. Thus, we made the decision to create a prototype that was as fully-featured as possible, rather than a 'smoke and mirrors' approach.

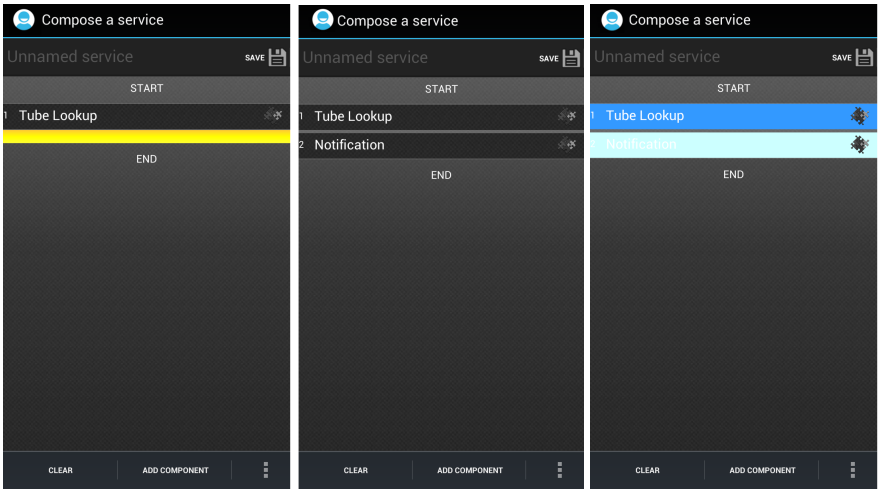
Based on the features that are supported by currently available EUSC applications, we decided that the minimum functionality of the prototype would be to allow the user to discover components, compose them to create a composite, and execute this composite and identify changes that they might want to make to it. This gives participants the chance to see each of the main stages of the EUSC life cycle that they would be expected to take part in if they were using an EUSC application of their own volition. Combining this minimum functionality with the preliminary requirements, we identified the following three main areas of functionality for the prototype:

1. Viewing, discovering and interacting with a selection of components.
2. Composing these components to create a composite (notably the composite described in the motivating scenario).
3. Viewing, discovering and interacting with composites.

Within each of these sections, we made various design choices based on the preliminary requirements, as well as reviewing tools that are currently available in the domain. An example of this was a requirement that stated that templates should be used within composition in order to make the process simpler for the user [Mehandjiev et al., 2010b]. In a review of available EUSC applications which we performed whilst creating the demonstrator specification, we identified a common template that is used across IFTTT, Zapier, and AutomateIt: *"if [Trigger] then [Action]"*. Another less restrictive template we were able to identify was simply a linear approach, i.e.: *"[Component 1] then [Component 2] then ..."* (e.g. Automator or Tasker). Other design choices that were made at this stage were presented to participants within the study sessions.

We decided to create a mobile tool (specifically on Android) due to the myriad of different services that are available such as contextual services and pervasive services as well as the rising popularity of Android-based EUSC applications. In the rest of this section, we will refer to the prototypical demonstrator application as 'Composer'. As we discuss earlier, we elected to make Composer as fully featured as possible, and hence fully support composition with a small number of components.

Figure 3-1 shows how users perform composition in Composer. demonstrated for the composition process that reflects the simplest version of the composition process in the scenario with which users were presented (see Section 3.3.1). Figure 3-1 (a) shows the composition process after the user has added the Tube Lookup component to the composite they are creating – note the yellow warning label indicating potential data loss (since the Tube Lookup component is providing them with data that they are not using). Figure 3-1 (b) shows the same view with the Notification component also added to the composition – the



(a) Composition Step 1 (b) Composition Step 2 (c) Composite Test Execution

Figure 3-1: The composition process in Composer (originally shown in [Ridge and O'Neill, 2014]).

warning label has been removed since the notification component is using the data that was provided by the Tube Lookup. Figure 3-1 (c) shows the composite whilst it is being tested – light blue indicating the component is currently executing, and dark blue indicating the component was executed successfully. The components are added to the composite through a component selection screen, which is shown in Figure 3-2.

In the remainder of this section, we will give a brief overview of each of the main functions of the demonstrator as the same breakdown was used for the walkthrough of the tool within the study sessions.

Component Discovery The only component discovery mechanism provided by Composer was a (short) browsable list of components that they could use. This list was divided into two based on the location of the component in question: they could either be already on the user's device, or available for them to download to the device from the Web. This section of the application is shown in Figure 3-2

Device-based components were either provided as part of Composer, or were in other applications on the device but could be executed remotely by Composer. Components available on the Web were part of applications that were not currently installed on the user's device, but could be subsequently installed and have their contained components executed by Composer.

The user can choose any of the components from the list of available components, where they are taken to a page that provides them with more information about that Component (not pictured). This page provides a description of what the service does, so that they are able

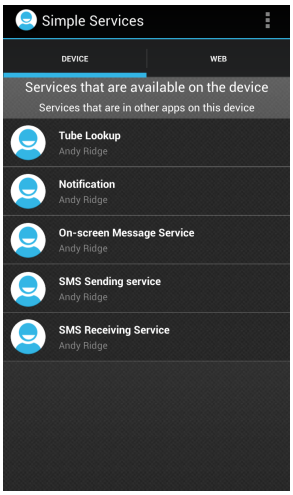


Figure 3-2: The list of available components in Composer (originally shown in [Ridge and O'Neill, 2014]).

to decide whether they wish to use it in composition or not. They can then choose to use it in composition, or in the case of components on the Web, they can download the application in which they are contained.

Composition The composition page (Figure 3-1) is the point at which the user can add components to the composition. Indicating that they wish to do this takes them to the component list, where they choose which component to add, which is then added to the list of components that are currently in the composition. Currently it is not possible to rearrange components in the composition, they must instead be removed and re-added in the correct order. Figure 3-1 and its associated description show the composition process to create the composite in the provided scenario, but the process is identical to combine other collections of components to create a composite.

On this page, users are able to test the current composition, once it has been verified as being correct (i.e. that data types are compatible with one another). The composition process can also be reset from this page. Once the user is happy with their composition, they can save it to create a new named composite.

Composite Interaction The final main section of Composer allows the user to interact with the composites that were created in the composition section of Composer. Composites are represented as a grid of icons, and there are four interactions that users can have with the composite: executing it, executing it on a timer (e.g. scheduling it to run each morning), editing the composite, and deleting the composite.

Scenario Generation

The main scenario that was used as part of the introductory materials for participants was:

"Ben has a London Underground tube line status service for his smartphone that allows him to check the status of any tube line. He feels that it's too much hassle to check each of these manually every time he needs to get the tube and wants his phone to notify him when there is a problem using the in-built notification service in the OS. There's no option in the service itself to do this, so he decides to use End-user Service Composition to fix his problem. Using the Composer tool, he is able to connect the phone's notification service onto the tube lookup service, so that when a problem is reported on a particular line (which he can choose), he will be notified via a new item in the notification tray."

"After using this service for a few days and being notified at strange times of day he decides that he wants to receive these notifications around the times he would normally be getting the tube. He chooses to edit the service, and adds the device's clock service in between the tube service and the notification service. He sets the clock to only let the notifications through between 6-8am, and 4-7pm."

The scenario was generated as part of a technical meeting with a number of experts in services and SC at a Mobile VCE Technical Steering Group at Glasgow University in August 2011. The motivating scenario presented in the introduction of this thesis is a minor evolution of the scenario developed at this meeting and presented above.

3.3.2 Data Analysis Strategy

[Sutcliffe and Ryan, 1998, Sutcliffe, 2003] provide little insight as to how requirements are elicited from the data that results from the SCRAM session, only that the video and audio output are 'analysed' alongside any notes taken during the session, and a final summary of the session with the participant(s).

One of the changes we made to the method was to move the requirements elicitation stage completely out of the study sessions (to prevent overloading participants – this will be discussed later), and as such we needed to decide upon a method for analysing the output of the study sessions in order to be able to elicit a set of concrete requirements.

Conventional (inductive) content analysis (i.e. grounded theory) is used in fields where little to no existing research exists, where the codes are identified directly from the data without imposing any structure on the data or the analysis [Hsieh and Shannon, 2005]. Thus, this approach relies on a less-structured, particularly open-ended interview approach [Hsieh and Shannon, 2005], and as such is not compatible with our method because participants are primed about certain topics within EUSC: those within Composer, or in the demonstration script and associated probe questions.

Instead, we opted for a more structured (deductive) approach: Directed Content Analysis (DCA). DCA uses prior knowledge in the domain to identify a set of initial coding categories [Hsieh and Shannon, 2005]. Then, when the transcript is being analysed, as topics or concepts are identified within these categories, a code is assigned to that data and is recorded [Hsieh and Shannon, 2005]. Any topics or concepts that do not fit within any of these initial categories are put to one side and analysed later in the process. Once the initial analyses is completed, the topics not found within the initial categories are coded, and grouped together into a set of subsequent categories.

DCA is particularly useful in interview situations where participants are asked about particular topics within a domain, as these topics can be used as initial categories, and can form the basis for the questions during the sessions [Hickey and Kipping, 1996]. Thus, this approach ensures that the results are robust to priming. Priming occurs in SCRAM because the participant is directed to answer questions about certain topics that are in the probe questions or reflected in the design of the prototypical demonstrator.

We used data from the initial requirements capture and from the domain review to identify a set of initial categories for our content analysis, which would also form the structure of the script and probe questions that is presented to participants. These categories were identified as those that participants will be primed to, and our analysis should be robust to this advanced knowledge. Some of these initial categories were identified from prior literature, others due to variations identified in currently available EUSC applications. The categories we identified were:

- **Composition flow**: References to the type of flow between components that is: (1) supported, and (2) represented to the user. This is normally either control flow or data flow [Mehandjiev et al., 2010b].
- **Composition – connections and compatibility**: References to the representation of composition to the user – how the components are connected to one another, how the user can determine if the components they have chosen are compatible with one another, etc.
- **Metaphor**: References to any metaphor that has been used to try to better convey the concept of SC to the user [Danado and Paternò, 2012].
- **Templates/examples**: References to templates and examples that can provide guidance in the composition process [Mehandjiev et al., 2010b, Picozzi, 2010].
- **Component type**: References to the 'type' of components that are supported by the tool, e.g. triggers in IFTTT.
- **Discovery/acquisition**: References to the life cycle stage at which point the user can discover new components to be used in composition [da Silva et al., 2008].
- **Attributes**: References to the attributes of components and composites that are presented to the user throughout the process [Picozzi, 2010].
- **Services**: Any other references to components or composites that were not covered in

any of the other categories.

- **Inputs and outputs:** References to how inputs to and outputs from components are supported by the EUSC application, and how it conveys this information to the user.
- **Testing:** References to the user being able to test the composite that they have created.
- **Aesthetics:** References to the visual nature of the composition process.

Any references that participants made that did not fit within this set of initial categories related to topics that participants had not been primed to, and were organised into a set of subsequent categories.

Adapting SCRAM

[Sutcliffe, 2003] makes a number of recommendations as to how SCRAM could be improved if it were to be used in future. These improvements are based on a number of weaknesses that they identified with the initial specification of SCRAM.

The first issue encountered by [Sutcliffe and Ryan, 1998, Sutcliffe, 2003] was the presence of bias towards the features or design choices that were made in the prototypical demonstrator over those made in other applications in the domain. That is, participants were more likely to suggest requirements relating to the design decisions that were made in the prototypical demonstrator over other design choices that could be made. To remedy this, they suggest presenting participants with sketches of alternative designs that could have been used for the demonstrator. However, we felt that this alternative would still include bias, since the design sketches are unfinished, and participants may think of these as less important *because* they were not chosen. Instead, we decided to present participants with examples of currently available EUSC applications when discussing different design decisions that were not chosen as part of the demonstrator (these are listed later).

SCRAM originally used Questions, Options and Criteria (QOC) [MacLean et al., 1991] to represent the various design decisions that were considered in the design of the demonstrator, and the various alternatives to the decisions that were made. However, [Sutcliffe and Ryan, 1998] found this to be ineffective as participants did not understand this representation, and suggested using tables instead. To remedy this problem, we decided it would be more effective to incorporate the discussion of different design alternatives whilst presenting the other EUSC applications.

Before finalising the procedure that would be carried out in our SCRAM sessions, we carried out three pilot sessions in order to assess the changes that we made to method, and to see if there were further alterations that might be beneficial. All three sessions ran for over 2 hours, which became fatiguing for participants and the effectiveness of discussions was reduced. Session overloading was identified by [Sutcliffe and Ryan, 1998] as a problem that can occur, and needed to be minimised. Since we could not remove any of the sections of the sessions without compromising the integrity and utility of the requirements that would be gathered.

Thus, we decided it was necessary to remove the requirements elicitation from the sessions themselves, and instead perform this *post hoc*.

The final alteration that we made to the sessions was to include only a single requirements engineer. We ran one of our pilot sessions with multiple engineers, and the other two with a single requirements engineer since we found that the second engineer was largely redundant due to the lightweight nature of the demonstrator. Reducing the number of requirements engineers also meant that we were able to reduce the number of participants per session, as the sessions were still balanced between participants and requirements engineers, meaning that participants could still retain ownership of the session [Sutcliffe and Ryan, 1998].

3.3.3 Method

Study Procedure

Study sessions were split into three parts. First, participants were presented with a set of introductory materials including an introduction to SC, a glossary of terms that might be used throughout the session (to which they could refer at any time), and a set of demonstrative scenarios to convey how SC can be used in real world situations. The second stage was the main body of the session, which followed the script presented in Appendix A.6). This was the section where participants were presented with the concept demonstrator and alternative applications, before being asked probe questions (listed in the demonstrator script) about various aspects of what they had seen. Finally, participants were invited to give any comments they might have on anything they had encountered in the sessions, before being presented with a questionnaire to assess their experience with SC and other related aspects prior to the session.

The aim of the introduction to the session was to provide participants with enough information to ensure that all participants had enough knowledge about SC to allow them to understand the sessions and be able to give insight and opinions on the scenarios and applications with which they were presented. After being provided with information about what SC is, they were given scenarios to give context as to how EUSC could be used in the real world, by ordinary people.

The main section of the sessions was the run-through of the script (Appendix A.6), which guided participants through various tasks in the concept demonstrator and other EUSC applications, before asking them various questions based on the topics identified prior to the sessions.

The other EUSC applications that were demonstrated to participants in the SCRAM sessions were: Tasker (Android), On{X} (Web & Android), IFTTT (Web), Yahoo! Pipes (Web), Automator (OSX), and Quartz Composer (OSX). These applications were chosen from those that were available at the time, and were meant to provide participants with a wide range

of design choices that have been made before. The EUSC applications demonstrated to participants in this study are described in Section 2.7.

This main part of the session was split into three sections based on the three main areas of functionality of the prototypical demonstrator that we identified earlier:

- Viewing, discovering and interacting with components.
- Composing these components to create a composite.
- Viewing, discovering and interacting with composites.

For each of these three sections, we performed the following set of tasks:

- Demonstrated that section of the prototype, following a scripted sequence. The same task was performed across all participants.
- Performed a similar task in each of the other EUSC applications. The same task was performed across all participants, although given the variety of EUSC applications chosen, the same task could not be performed in each application.
- Interviewed the participant using the set of questions identified for that part of the application, using the prompts only if necessitated by a lack of response or understanding by the participant.

Finally, we gave a questionnaire to assess participants' prior experience with SC, their technical skill, and their knowledge of the domains they would encounter in the sessions (travel, weather, mobile messaging). We decided to give participants the questionnaire at the end of the sessions so that their thought processes in answering the questionnaire did not influence them within the main body of the session.

Each session lasted approximately 90 minutes and was video recorded.

Participants

We performed 10 sessions, each with 1 participant: 5 male and 5 female. Since sessions were reasonably long and a large amount of data were gathered in each session, we felt that 10 was an appropriate amount. Our participants had a mean age of 27.8 (SD = 10.18), 5 were students, 4 employed, and 1 self-employed. Participants' backgrounds included Computer Science (3), Physics (2), Beauty (2), Engineering (1), Psychology (1), and Geography (1). None had prior experience with SC.

Due to the semi-technical introduction to EUSC at the start of the session, we restricted participants to those who currently owned smartphones to ensure that they would understand the concepts involved in the introduction and the explanatory scenarios. They were recruited using posters and all were smartphone owners (5 iPhone, 4 Android, 1 BlackBerry).

Materials

The materials that were given to participants were as follows (all were paper-based):

- Consent form detailing the study and requesting consent to be video recorded.
- Questionnaire (demographics, technical expertise, SC knowledge, domain knowledge).
- Introduction to the session, detailing the overall goal of the session and explaining how it would be carried out.
- Glossary of terms defining the terminology that would be used in the session.
- A collection of illustrative scenarios including the main scenario which is given in Section 3.3.1

The demonstrator script and associated probe questions was retained by the requirements engineer throughout the session. The requirements engineer used a Macbook Pro that was set up to demonstrate the desktop and Web-based example applications. An Android smartphone was used to demonstrate Composer and Tasker – the example EUSC applications that run on the Android platform. A camera was set up in the room to record a video of the sessions which could be later analysed to derive requirements.

3.3.4 Post Study Method

Since one of our changes of the original SCRAM procedure was to remove the requirements elicitation stage from the study sessions, we instead performed this post hoc. Section 3.3.2 discusses the different options that were considered before we opted to use Directed Content Analysis (DCA).

The first stage of DCA is to generate a set of categories into which codes identified in the data are categorised [Hsieh and Shannon, 2005] (our initial categories are listed in Section 3.3.2). Coding was then completed by working through the transcript and identifying when participants discussed topics that could then be classified as codes (i.e. they represented topics within the domain). If the code belonged within one of the initial categories, it was recorded, otherwise it was put to one side. Once the initial coding had occurred and all codes from initial categories had been identified, the leftover identified codes were organised into a set of subsequent categories that were generated in an ad hoc manner.

It is important to note that when performing DCA, ranking and frequency comparison should be used over statistical testing for significance to ascribe relative preference between the codes and categories [Curtis et al., 2001]. However, it is important take into account the priming towards the topics in the initial categories.

3.4 Results

The study yielded a set of 139 requirements for an EUSC application, which are listed in Appendix C. The process for deriving these requirements relied first on content analysis to extract topics and concepts from the session transcripts in the form of codes. We then analysed these codes to elicit the requirements. This section discusses the codes that were identified from the transcripts, and the next section describes how these codes were transformed into a set of requirements. Note that *codes* are always in italic, and **categories** are bold.

Within each of our initial categories were a number of codes, and along with each code we recorded the total number of occurrences of that code across all participants (**O**), and the number of different participants who identified that code (**P**). For example, within the category "Attributes (components)" we identified the codes *Description* [24O, 5P], *Number of uses* [8O, 5P], *Cost (free)* [8O, 5P].

In total, we identified 188 codes, 157 of which were identified within initial categories, and 31 of which were not. As part of DCA, codes that did not fit within any of the initial categories (those identified prior to the study, listed in Section 3.3.2) need to be organised into another set of subsequently identified categories. We did this in a bottom-up manner, grouping together codes that were related to one another, adding to that group until no more codes related to the topic that the subsequent category represented. We identified the following subsequent categories:

- **App feature**: References to general features of EUSC applications.
- **Social**: References to the connections that the EUSC application allows the user to have with their friends, or other users the the application (normally relating to social media).
- **Assistance**: References to the assistance provided by the EUSC application to the user.
- **Specific app or function**: References to specific aspects of EUSC applications that participants were presented with in study sessions.
- **Accessibility**: References to accessibility features provided by the EUSC application.
- **Comparison with non-SC tool**: Comparisons that participants made between EUSC applications and applications within other domains.

To compare the relative popularity of the codes that we identified, we used quantitative comparisons to apply a rank/ordering to the codes identified within the data [Curtis et al., 2001]. Whilst these comparisons are useful to help with our discussion of the codes, we must be aware of the priming that participants had for the codes within the initial categories, and that the quantitative comparison approach is very simplistic.

In particular it does not make sense to compare the relative popularity of a code from an initial category with a code from a subsequent category, because participants were primed about topics represented by codes in initial categories, and thus more likely to refer to these

in their responses throughout the study.

Tables 3.1 and 3.2 show the 10 most popular codes across participants in each of the initial and subsequent categories, respectively. The occurrences of each code were only identified if the participant made an explicit reference to that topic that the code represented. This meant that the cases where participants were asked about the topic were not recorded.

Table 3.1: The 10 most popular codes identified within initial categories (P = # participants, O = # overall).

Code	Description	P	O
Input/output	Inputs and outputs of components.	9	28
Examples	Examples of components used in composites.	9	15
User ratings	Some user ratings for services (e.g. star ratings).	8	32
Group by function	Grouping services by their function.	8	14
Name not representative of function	The name of the component isn't representative of the function it performs.	8	14
Metaphor	The use of metaphor to convey SC concepts to the user.	7	10
Triggers vs. other components	Comments on the difference between triggers and other components in the application.	7	10
Explicit connections between components	The representation of connections between components is explicit.	7	12
Testing of composition	The user should be able to test the composites that they create.	6	7
Grouping or splitting by component location	Components should be grouped together based on where the user can find them.	6	6

This method of ranking the codes by frequency of use shows which codes were used across participants, which we felt was a more important measure than the number of times the code was used overall. Comparing popularity across participants from the initial and subsequent categories shows the difference in popularity with or without priming: of the top 10 most popular codes in initial categories, all were referred to by more participants than any of the top 10 codes from subsequent categories. A simplistic requirements elicitation approach might only consider the most popular codes as candidates from which to derive requirements, whereas there are other codes that present other interesting findings of this study, given some of the more interesting topics were only derived from a single participant. We identified these interesting codes in both initial and subsequent categories based on concepts that we believed would advance the domain. A subset of these 'interesting' codes are:

- *OS integration*: The EUSC application should be able to integrate with functions built in to the operating system of the device, for example through home screen interactions with widgets or shortcuts.
- *Composing pervasive services*: The list of components that can be composed should include those that can be discovered in the environment. This feature is now available

Table 3.2: The 10 most popular codes identified in subsequent categories (P = # participants, O = # overall).

Code	Description	P	O
Components – input-only vs. output-only	The difference between components that only have an input and those that only have an output.	5	10
Terminology confusion	The participant was confused by the terminology used in the tool.	5	8
Choosing components is like choosing apps	The component selection process is similar to the app discovery process.	5	5
Like	The participant liked some aspect of the tool.	4	12
Two versions of tool – high-tech and low-tech	Different user technical abilities require different levels of support from the tool.	4	7
Warnings	The tool warns users of potential errors in the composition.	3	6
Size of composition – mobile vs. desktop	Different aspects of composition based on the size or context of composition.	3	6
Listing composites is like listing apps	The process of interacting with composites is like interacting with apps on a phone.	3	5
Assistance provided in the tool	The assistance that is provided to users while they are using the tool.	3	4
Composites need more information	Users wanted to be able to find out more information about composites.	3	3

in one of the presented EUSC applications (IFTTT), although this feature was not available at the time the study was carried out.

- *Making a description for a composition means you can make a composition from a description*: Composition should be able to occur automatically based on a description of the required composite presented by the user. Participants were not introduced to the idea of automatic composition through either the introductory materials or the other EUSC applications.
- *Automatic composition identification*: The EUSC application should "watch" the actions that the user performs regularly, and be able to identify when SC could automate one of these tasks for them.
- *Infinite composition*: Composites should be usable as components. As with automated composition, this topic was not presented to participants in either the introduction or the example EUSC applications.

3.4.1 Participant Demographics

In the demographics questionnaire, participants were asked to indicate if they had experience with programming or software development so that we could distinguish between the requirements that were provided by participants with differing levels of technical knowledge or skills. 5 of our participants had experience with software development, and 5 did not. Of

the 188 total codes, 78 were identified by both groups, 68 by developers only, and 42 by non-developers only.

3.5 Requirements Derivation

This section describes the process that was used to analyse the codes that were gathered in the study sessions and generate a set of requirements for EUSC applications. To demonstrate how this was achieved, we walk through the process for a small, representative set of codes that illustrate the elicitation process. The full set of requirements is available in Appendix C. Codes were one of four different types, and we will demonstrate the analysis process with some codes from each of the following four types:

T1. Codes relating to topics found in prior work on EUSC that has been validated by our work, e.g. control flow and data flow [Mehandjiev et al., 2010b].

T2. Codes that relate to concepts that are identifiable in the existing EUSC applications presented to participants, but not present in prior work on EUSC, e.g. attributes, component types.

T3. Codes that relate to concepts from other domains that are not present in existing EUSC applications presented to participants, e.g. user ratings for composites.

T4. Codes that relate to new ideas suggested by participants, e.g. 'infinite' composition, pervasive composition, automatic composition identification.

T1 One of the points during the session at which participants would either identify *Control flow* or *Data flow* was when they were asked to describe how they would interpret a particular representation of the composition. This was meant to assess whether they would associate it with control or data flow, in a similar way to Mehandjiev et al. [Mehandjiev et al., 2010b]. Participants' responses included: "The tube is looking up stuff and then it's notifying you *and then* it's the end." – P1; "The tube component *passes something* to the notification" – P6; "You also need to understand how the *data* moves in this on" – P2. Codes were assigned to these responses based on participants' focus on either the order of execution of the components, or their awareness of the data being passed between the components. Our analysis of these codes and responses yielded two requirements that had been identified previously by Mehandjiev et al. [Mehandjiev et al., 2010b]:

R1. The flow of control between components should be represented in composition.
The application should present the order in which the components are executed.
Rationale: Users need to be able to identify the order in which components are executed in the composition.
Category: Non-Functional – Operability – Appropriateness recognisability
Criticality: High
Risk: Low

Source: [Control flow]
Prior identification: [Mehandjiev et al., 2010b]
COTS[16]: Atooma, Tasker, AutomateIt, IFTTT, Zapier, Automator

R2. The flow of data between components should be represented in composition, if it is present.
The application should show how the data is passed between the components in the composition.
Rationale: Users should be able to identify what data is being passed between components in the composition.
Category: Non-Functional – Operability – Appropriateness recognisability
Criticality: Medium
Risk: Low
Source: [Data flow]
Prior identification: [Mehandjiev et al., 2010b]
COTS: Yahoo! Pipes, Quartz Composer

T2 The second type of code that we analysed related to concepts that were present in the design of the EUSC applications with which participants were presented during the study, but had not been identified in prior work. The example codes that we will discuss here are *Testing* and *Sharing*. *Testing* relates to the user being able to test the composite that they are creating at various points throughout the composition process to ensure that it runs correctly, *Sharing* relates to any features that allow the user to share the composites that they have created with other users of the application, and conversely discovering the composites that have been shared by others. Neither of these codes were identified in our preliminary requirements.

R3. The application should allow users to share services.
The application should allow users to share services that are created using the application with other users of the application.
Rationale: Once a user has created and used a composite, they might want to share it with others.
Category: Functional – Management – Monitoring & Adaptation
Criticality: High
Risk: Low
Trigger: The user indicates they want to share the composite.
Preconditions: There is a composite to share.
Postconditions: The composite is shared to a shared composite repository.
Failure effects: The composite is not shared.
Associated requirements: **R18**,**R23**.
Source: [Sharing/publishing of composites]

[16]Commercial Off-the-shelf Software

COTS: Atooma, IFTTT, Yahoo! Pipes, AutomateIt, On{X}

R4. The application should allow users to test composites.

The application should allow users to test execution of their composites-in-progress whilst they are creating them.

Rationale: Users need to ensure that composites they create function as intended.

Category: Functional – Verification

Criticality: Critical

Risk: Low

Trigger: User indicates that they want to test the composite.

Preconditions: The composite contains some components.

Postconditions: The composite executes in its current state.

Failure effects: The component does not execute as intended.

Associated requirements: **R15**,**R17**.

Source: [Testing]

COTS: [Yahoo! Pipes, Quartz Composer, Automator, Tasker, Yahoo! Pipes, On{X}

T3 The third type of code that we analysed related to concepts that were not part of the design of the EUSC applications presented to participants, but are present in the design of applications in other, similar domains. The example we discuss from this set of codes is *User ratings*, where users would be able to rate composites created by other users, or rate components that they have used in composition so that other users can use them as a quality measure. Participants felt that this would be a good addition, notably for composites: ‘*‘But obviously having ratings for them all would be quite cool too”* – P1. *“But then ratings would be more important for composites”* – P5.

R5. The application should allow users to rate services.

Users should be able to rate services to convey their opinion on the quality of the service to other users of the application.

Rationale: Users should be able to provide feedback to the creators of components and composites.

Category: Functional – Management – Monitoring & Adaptation

Criticality: Medium

Risk: Low

Trigger: The user indicates they want to rate the service.

Preconditions: There is a service to be rated.

Postconditions: The service’s current rating gets aggregated with the new rating.

Failure effects: The rating is not applied.

Associated requirements: **R25**

Source: [User ratings]

COTS: AutomateIt, IFTTT, Yahoo! Pipes, On{X}

T4 The final type of code that we analysed are those that we consider as new concepts that the participant is unlikely to have had exposure to elsewhere. These were often the topics identified in our discussion of interesting codes in Section 3.4. The example of this type of code we consider is *Automatic composition identification*: *"It would be quite cool for it to be able to identify things for you that you might not think about automating. Like for examples if it watched things you do and suggested compositions for you."* – P5. This yielded two further requirements:

R6. Potential compositions could be identified automatically.
The application should be able to monitor the activities of the user and identify tasks that they perform regularly that could be adapted to form a composite.
Rationale: If the application were able to automatically identify potential compositions, it would reduce user burden in deciding what compositions to create.
Category: Functional – Specification
Criticality: Low
Risk: High
Trigger: The user performing a manual task repeatedly.
Preconditions: The application is installed on the user's device.
Postconditions: A composite is created that performs the repetitive task.
Failure effects: The composite is not created.
Associated requirements: None
Source: [Automatic composition identification]

Using these four types of code as the basis for our analysis highlights a further category: requirements identified in prior work whose topics were not present in any of our codes. The only example of topics that fit within this category is the security of services. This highlights a deficiency in our set of requirements, which we address below in our discussion.

After finalising our analysis, there were 7 codes leftover that we were unable to derive a requirement from. The leftover codes were not translated into requirements because either they were a generic positive or negative comment about something, or they were far too specific, i.e. a reference to a single component within an EUSC application.

3.6 Discussion

Our study yielded 139 requirements that needed to be organised into a useful grouping. After creating the whole set of requirements, we split the requirements into functional and non-functional requirements, and then grouped these into sub-groups based on the results of our model-based validation of our requirements, which is described in Section 3.6.1.

A large proportion of the requirements we gathered could be used as they are in the specification of an EUSC application. A smaller proportion would require further research before they

could be used in the requirements specification for an EUSC application since it would not be clear that the requirement could be achieved effectively with the current state of knowledge in the EUSC domain. Three of the codes associated with requirements that require further research are: *Automatic composition identification* and *Automatic composition generation from description*. Below, we present a discussion of why these code requires further research (the required research is out of the scope of this thesis):

- *Automatic composition identification*: One of our participants suggested that the EUSC application could 'watch' tasks being performed by the user, and the application would be able to automatically generate a composite that automates the task the user was undertaking. This is an interesting feature for an EUSC application to have, but prior research has recommended against high-levels of automation in EUSC [Vulcu et al., 2008], and thus more research in this area is required.
- *Automatic composition generation from description*: Given a description of a composite, the EUSC application should be able to automatically generate a composite that meets this description. This is a typical example of automated composition, which is not recommended within EUSC [Vulcu et al., 2008].

3.6.1 Requirements Evaluation

Before we could be confident that our requirements could be used to motivate the design of an EUSC application, our set of requirements as a whole first needed to be evaluated. The requirements could be evaluated practically by using them in the design of an EUSC application, but to be able to incorporate the requirements into a design space for EUSC, we decided to perform an evaluation of the set of requirements in isolation. We evaluated three properties of our set of requirements: completeness, correctness, and consistency, which were identified as being important by [Zowghi and Gervasi, 2003].

Completeness

The first property of our requirements that we evaluated was their completeness – how complete they were. Incompleteness of requirements is a problem that has been identified as one of the more common causes of accidents and failures in systems [Zowghi and Gervasi, 2003], and furthermore, is difficult to identify in a requirements specification [Zowghi and Gervasi, 2003].

There are a number of methods for measuring requirements completeness, including: model-based evaluation, individual evaluation, requirements metadata, and the creation of a requirements specification document [Firesmith, 2005]. We performed individual evaluation and model-based validation to ensure completeness of our requirements. We chose individual evaluation since no additional resources were required, and model-based evaluation since

our work on the life cycle of EUSC provided us with a model against which the requirements could be evaluated.

Various different types of model can be used to evaluate the completeness of functional requirements, including data models and process models. [Firesmith, 2005]. In the literature review (Section 2.4), we discussed two process models (described as life cycles) created for EUSC [da Silva et al., 2009] and Service-oriented development [Mehandjiev and De Angeli, 2012]. We evaluated our requirements against an amalgamation of these models, utilising the relative strengths of either model. We recorded the number of requirements that were classified within each stage of the process model. Figure 3-3 shows the distribution of the requirements across each of the stages of the process model against which they were being evaluated, along with established models for non-functional requirements.

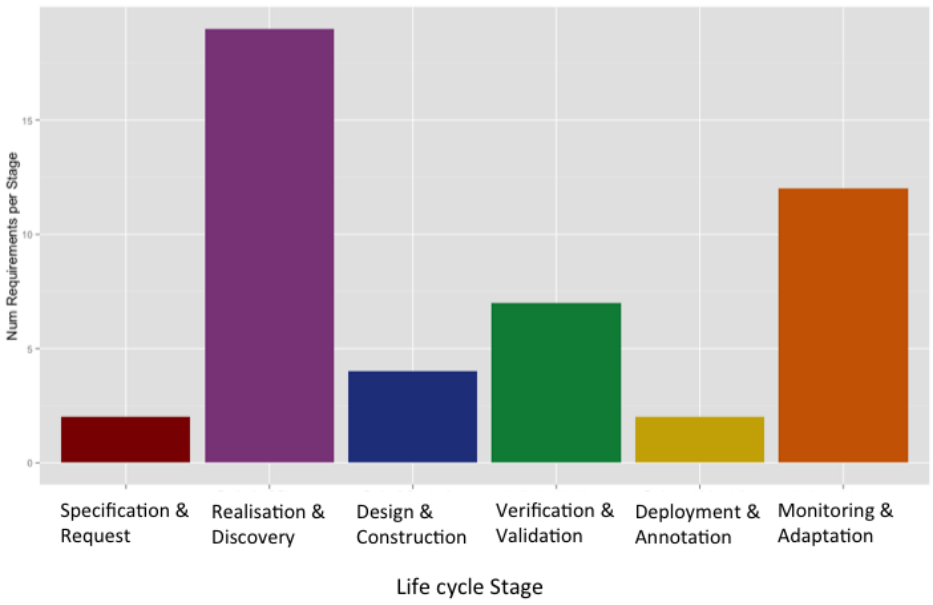


Figure 3-3: Distribution of requirements across the stages of a simplified EUSC life cycle process model.

All of the stages of our process model contained requirements, although the distribution of the requirements across these stages is not uniform. We believe that the inconsistent distribution across the different stages of the model is based on user involvement in each stage, and hence their perception of its importance in the overall process. For instance, the specification/request stage is something that we highlighted as being more relevant to automated composition, and as such not as important in EUSC. It is also difficult to separate from the discovery stage since without automation it is hard to see why a user would be specifying what they wanted without discovering components that would help them do so. The deployment and annotation stage of the life cycle also contained few requirements,

which we believe was due to one of three reasons:

- The annotation and deployment section of the life cycle was not identified as being within initial categories and hence was not a topic of any the probe questions, i.e. participants were not primed.
- None of the EUSC tools presented to participants implemented explicitly presented this stage as being separate from performing composition.
- This stage would typically have little involvement from the user's perspective.

The two requirements that were identified as being part of the annotation and deployment stage were both elicited from comments that participants to which participants were unprompted. We believe that the model-based evaluation demonstrates that our functional requirements are complete with the caveat that this completeness is from the user point of view, since all stages of the life cycle in which they are directly involved are considered, and those which are not have limited end-user involvement.

Following the evaluation of our functional requirements against the life cycle process model, we evaluated our non-functional requirements against the quality model presented in ISO/IEC 25010 [ISO/IEC, 2008] as it is an established way of classifying non-functional requirements. The quality model within ISO/IEC splits non-functional requirements into 12 categories, each with associated sub-categories [ISO/IEC, 2008]:

1. **Functional suitability**: appropriateness, accuracy, functional suitability compliance.
2. **Reliability**: Availability, fault tolerance, recoverability, reliability compliance.
3. **Performance efficiency**: time behaviour, resource utilisation, performance efficiency compliance.
4. **Operability**: appropriateness recognisability, learnability, ease of use, helpfulness, attractiveness, technical accessibility, operability compliance.
5. **Security**: confidentiality, integrity, non-repudiation, accountability, authenticity, security compliance.
6. **Compatibility**: replaceability, co-existence, interoperability, compatibility compliance.
7. **Maintainability**: modularity, reusability, changeability, modification stability, testability, maintainability compliance.
8. **Transferability**: Portability, adaptability, installability, transferability compliance.
9. **Quality in use**: usability in use, flexibility in use, safety.
10. **Usability in use**: effectiveness in use, satisfaction in use, usability in use compliance.
11. **Flexibility in use**: context conformity in use, context extendability in use, accessibility in use, flexibility in use compliance.
12. **Safety**: Operator health and safety, public health and safety, environmental harm in use, commercial damage in use, safety compliance.

When we evaluated our requirements against the contents of this model, we found that that vast majority – 76 requirements – were identified as relating to operability, and 6

requirements related to maintainability. No requirements were classified within any of the other categories within this quality model. 12 of our non-functional requirements could not be classified within any of these categories. Figure 3-4 shows the distribution of the non-functional requirements within the sub-categories of operability. In the maintainability category, 5 requirements fit within changeability, and 1 fit within modularity.

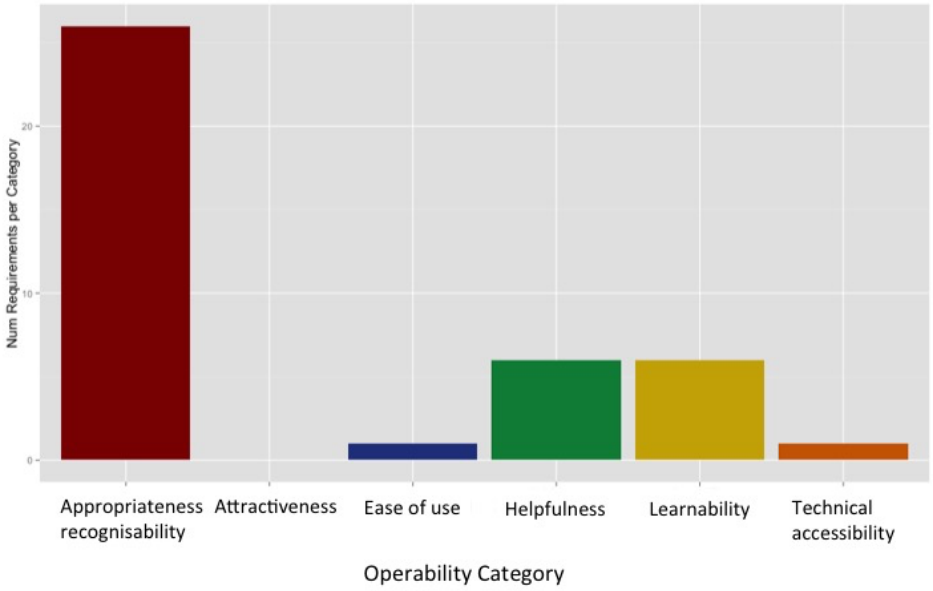


Figure 3-4: Distribution of requirements within the non-functional operability category.

We believe that the operability category was a focus of our participants due to its focus on topics that relate directly to using the EUSC application, as opposed to aspects of quality which users might not directly associate with their use of such an application, e.g. safety or transferability. Non-functional quality metrics were also not within any of the topics that were covered by the probe questions that were part of the demonstrator scripts.

This shows that our non-functional requirements are not complete, since there are a number of categories in the quality model in which there are no, or few of our requirements. This highlights a deficiency of the method that we used to derive the requirements, which is discussed further in Section 3.8.

We grouped the remainder of the non-functional requirements that we could not group within the quality model from ISO/IEC 25010 using a similar method to the grouping of the codes within subsequent categories (described in Section 3.4). We identified three categories: architectural requirements (7), representation/interface requirements (4), and interaction requirements (1).

The final step of our evaluation of the completeness of requirements to ensure that each

requirement was complete by performing individual evaluation against various properties that the requirement should have to be considered complete. An individual requirement is defined as being complete if it contains all the information that is necessary for it to be unambiguous and require no modification in order to be able to be implemented and validated [Firesmith, 2005]. The attributes to be associated with each complete requirement were [Firesmith, 2005]:

- Rationale
- Source
- Category
- Criticality
- Risk
- Prior identification
- Associated requirements
- Software that implements that requirement
- Clashes or conflicts with other requirements

Additionally, for the functional requirements we also ensured that the following information was associated with each requirement: trigger, preconditions, postconditions, and failure effects. The full list of the requirements and the information associated with them is shown in Appendix C.

The individual evaluation ensured that each of our requirements was individually complete, although the model-based verification indicated that the non-functional requirements were not complete. The implications of this will be discussed in Section 3.8.

The categorisation process used here has some similarities to the work on design spaces, which is discussed in Chapter 4, particularly when we consider the difficulty of parsing a list of 139 requirements. The link between design spaces and requirements has previously been identified by [Geyer, 2000], who indicates that design spaces can be a useful method for presenting requirements. These requirements form an important part of the design space that is presented in Chapter 4.

Consistency

To ensure that a set of requirements is consistent, one must ensure that there are no clashes or conflicts between them. In particular, there should be no contradictions between any two of the requirements [Zowghi and Gervasi, 2003]. Consistency of requirements may also refer to the consistent use of terminology throughout the set of requirements as a whole. To evaluate our requirements for consistency, we iterated through the requirements and sought to identify any conflicts or contradictions that existed between them.

We identified a conflict between requirements that related to the mechanisms that the EUSC application could use to group the different types of service that it presents. The first requirement stated that requirements should not be grouped (**R5.8**), and other requirements specifying how requirements should be grouped (**R5**, **R5.1**-**R5.7**). Addressing this conflict

required a different view of the former requirement stating that grouping should not be allowed. Instead, this requirement should be considered as another option for how grouping should occur rather than being a separate antithetical requirement. That is, the EUSC application should allow the users to view services grouped based on a number of different metrics, one of which is that the services are un-grouped. This solution to the conflict agrees with one of our other requirements, which states that the metrics that can be used to group services are customisable (**R60**).

Another area in which we identified conflicts between requirements was in the EUSC applications propensity to restrict the composition process or allow freedom to the user in this process. One requirement stated that composition should be unrestricted (**R35**), and a conflicting requirement that stated that the process should be restricted somehow by the EUSC application (**R59**). Our solution for addressing this conflict was identified from within another requirement that related to the use of templates within the EUSC tools (**R34**). This solves the conflict because the EUSC application could provide a template that would restrict the composition, or give the user a 'template' that does not restrict the composition process in any way. Providing templates also solves another potential conflict between the composition process being both simple (**R50**), and yet also comprehensive and complex (**R11**). Templates with varying complexity mean that some users could use simple templates to perform simple tasks, and others could use complex templates for complex tasks.

Whilst we have identified a number of instances of inconsistency between some of our requirements, in each case there was another requirement that resolved this conflict. Thus, we can conclude that overall, our set of requirements is consistent.

Correctness

[Zowghi and Gervasi, 2003] define correctness of a set of requirements as the combination of the requirements' completeness and consistency. Another more practical viewpoint is that requirements are considered to be correct if they map directly onto the needs of the users of the system to be created based on the requirements. In our case it is much more difficult to use a definition of correctness that relates to the needs of 'a user', since we do not have a single business customer, and the needs of our participants and hence the eventual users of the EUSC application are likely to be very varied. As such, it would be difficult to use these needs to determine whether requirements are 'correct'.

We were also unable to evaluate the correctness of our requirements with the participants in our sessions since we altered the method to move the requirements elicitation process after study sessions. Given the varied backgrounds of our participants, we also felt that it was unlikely that there would be a consensus between all the participants as to whether requirements are correct.

Our method for evaluating correctness of individual requirements was to identify whether

these requirements had been used before, either by being present in a set of prior EUSC requirements, or by having already been implemented in an EUSC application. An evaluation of the requirements showed that roughly half were visible in currently available examples of EUSC applications. In the appendix, each requirement lists the EUSC applications in which it has been identified under the heading 'COTS'.

To individually evaluate the remaining requirements, we used peer review [Westfall, 2009]. We invited researchers who had prior experience in the domain of SC and EUSC to individually evaluate each of the requirements, none of which were identified as being incorrect.

3.7 Method Generalisability

Finally, we will discuss the generalisability of our chosen requirements gathering method. The method that we used was was based on an existing requirements gathering method that is already generalisable: SCRAM. In the rest of this section, we will break the method down into its constituent stages, and discuss how the method can be generalised at each of these stages. An overview of the stages in the method are shown in Figure 3-5.

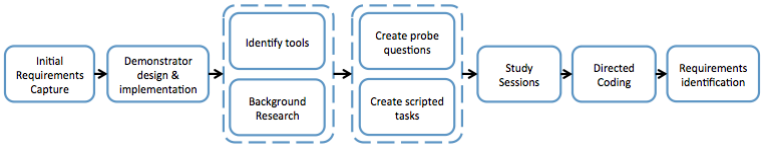


Figure 3-5: The stages of our adapted requirements gathering method (originally shown in [Ridge and O'Neill, 2014]).

The changes that need to be made in order to carry out the preliminary requirements capture in a new domain depend greatly on the resources that are available in that domain from which these preliminary requirements can be derived. In our case, we were able to use the results of prior requirements gathering work in EUSC. However, this may not be feasible for other domains given that prior requirements gathering literature may not be available. The only other domain resources used as part of our method were existing EUSC applications, which were reviewed in order to gather requirements. Clearly this could be applied to any other domain in which applications already exist. If the method is being used in a domain where neither of these resources are available, then there are recommendations suggested by Sutcliffe and Ryan [Sutcliffe and Ryan, 1998] that were part of the original version of SCRAM that can be used instead. One such recommendation was to use questionnaires to gather an initial set of requirements.

Once the preliminary set of requirements has been gathered, the design and implementation of the prototypical demonstrator can be carried out using any standard method that the requirements engineers deem suitable.

The next set of tasks to be completed are to identify the resources that are needed for the study sessions, and to generate the demonstrator script and probe questions. Existing applications in the domain need to be identified to be demonstrated to participants, which can also be used as part of the research for topics on which the probe questions are based. If there are no suitable applications in the chosen domain, then applications from related domains should be used.

The study sessions themselves should operate in the same way that is reported in Section 3.3, since the chosen domain would not make a difference to the way that these sessions run.

After the sessions are completed, the two remaining stages can both be carried out as they are reported in Section 3.3, since these stages are simply a method that is applied to the resources that have already been gathered as part of the method. That is, the output of the study is transcribed, and then coded using DCA and the set of initial categories identified prior to the study being carried out. Finally, requirements elicitation can be carried out by analysing these codes.

3.8 Limitations

The most obvious limitation of our work is that our non-functional requirements are not complete. There were a number of categories within the quality model in which we were either not able to classify any requirements, or we were only able to classify very few. Furthermore, of the categories in which we were able to classify requirements, they were not distributed uniformly. That is, the majority of our non-functional requirements fit within the category of operability, few within maintainability, and none within functional suitability, reliability, performance efficiency, security, compatibility, transferability, quality in use, usability in use, flexibility in use and safety. We believe that this limitation was due to a number of independent factors: our focus on consumers as end-users, and these topics being missing from our probe questions. We believe that as potential consumers, our participants were inherently less likely to consider aspects such as maintainability of their own accord than a developer would, and furthermore participants were not prompted with topics in these areas within the probe questions.

To address the lack of coverage of areas of non-functional requirements in future implementations of this method, we suggest that each session have a section that has a focus on the non-functional quality-based topics were not covered by our non-functional requirements. This ensures that participants are prompted about these topics, and hence should result in a more even distribution of requirements across the non-functional quality requirements categories.

Another limitation that is identifiable in our requirements is the difference in relative maturity of the requirements we gathered – some could be usable immediately to inform the design

of an EUSC application, whereas others would need a great deal more research before they could be used in this way. To mitigate this, we believe that further validation of the requirements is an important next step. This validation is achieved when the results from the requirements gathering study are included in our design space for EUSC applications since DSs have been suggested as being useful method for displaying requirements [Geyer, 2000]. The inclusion of the requirements into our DS is discussed in Section 4.3.

In the analysis of the codes that were derived from the data from the study sessions, we split the codes into four types based on whether they have been identified before, either in prior EUSC requirements, other EUSC applications, other applications in other domains, or whether they are a new concept. This structure suggested two other types that we did not consider:

- Concepts present in the designs of available EUSC applications that were not identified in our study.
- Concepts identified in prior requirements gathering efforts for EUSC that were not present in ours.

It is evident that there must be some concepts present in available EUSC applications that we did not gather as part of our set of requirements, since our set of requirements could not completely describe all of the available EUSC applications without encountering conflicts.

We have only been able to identify one concept that was identified in prior sets of requirements for EUSC applications but not within ours: security. Maintaining the security of services was identified as a requirement by [Namoun et al., 2010b], however it was not identified as being broad enough to be an initial category, and hence was not the topic of any of the probe questions within our SCRAM sessions. Furthermore, none of the demonstrative SC tasks carried out within study sessions involved any components with which users might have to associate their personal data. We believe that it is likely that if the user was required to enter any of their personal information (passwords, addresses, financial details, etc.), then they would have been more aware of security as a concept upon which requirements could be based.

The omission of security within our set of requirements highlights a limitation of the method that we used to gather them: priming. That is, participants were more likely to discuss the topics that were suggested to them (either directly or indirectly) by the probe questions throughout the sessions. Consequently, requirements were much more likely to be derived based on the topics from these questions, and less likely for any topics that were not included. It would be possible to mitigate this issue by analysing each session before running the next, and adding topics new topics identified in the earlier session to later sessions. This approach is not without issue however, since adding more questions would run the risk of overloading the participants of these later sessions. Furthermore, it would complicate the analysis as the difference between codes in initial categories and those in subsequent categories would be reduced.

The effect of priming is easy to see when we consider that 85.1% of codes were from initial categories, i.e. those topics that we identified as being important prior to carrying out the study. This indicates that different participants would have had little effect on the requirements gathered.

In the study sessions themselves, we could not perform the same task across each of the EUSC applications that were demonstrated to participants since even the EUSC applications varied greatly in their usage context (i.e. mobile, Web, desktop), and their target domains. We thought that this variation in task was acceptable since the variation in the design of the tools was more important than demonstrating the same task across each of the tools.

The method as a whole is demanding in time and resources, both during the sessions with participants and in the analysis that is carried out afterwards in order to elicit the requirements. After carrying out sessions with 10 participants, we had gathered enough data from which we were able to derive a large and robust set of requirements. We felt that this was enough because it minimised the duplication of content between participants, as well as ensuring that the time and resources required to analyse the data were both realistic and manageable.

The set of requirements we present here are meant to describe a single instance of an EUSC application, as well as the requirements being individually applicable to numerous potential EUSC applications. The generalisability of the individual requirements to EUSC applications across multiple domains and platforms highlights the variability in these applications, and we believe is a strong motivator for the use of design spaces to both describe existing EUSC applications and to help designers to create new ones.

3.9 Chapter Summary

This chapter addressed the first of our three research goals: **RG1**. **Derive and evaluate a set of requirements for an EUSC application.** We described a study to gather a large and robust set of requirements for an EUSC application using a generalisable and repeatable method. We were motivated by two small sets of requirements identified in prior work (described in Section 2.8) which identified a small number of topics that require further research in the domain. However, it is difficult to use such a small set of requirements as the basis for the design of an EUSC application, which prompted our approach.

We described the method used to gather the requirements, which is based upon an established requirements gathering method called the Scenario-based Requirements Analysis Method (SCRAM), and a data analysis method called Directed Content Analysis (DCA). We adapted, extended and combined these two method to create an end-to-end method for gathering requirements from end-users that is both repeatable and generalisable to other domains.

Next, we discussed the analysis the set of requirements for correctness, completeness and consistency and considered their validity. Finally, we described the limitations of the work

and how the method could be generalised to be used in domains other than EUSC.

Individually, the requirements we have gathered are applicable to a wide range of EUSC applications, and as such present viable alternatives within the design space for EUSC applications. Their identification as requirements in this instance does not mean that they would be *required* for every EUSC application, but most would represent a viable choice. In the next chapter, we consider the design space for EUSC applications, where one of the stages of creating this design space was to add the results of the requirements gathering work presented here.

This chapter has two contributions: a large set of requirements for EUSC applications, and an empirically tested method for establishing this set of requirements. These contributions match up with the two objectives of this chapter, meaning that we have addressed our overall aim.

The method that was used to gather the requirements for EUSC applications could be used to gather requirements for any other end-user focused domain, which we believe is particularly useful given that requirements gathering approaches have previously tended to be geared towards business customers.

The requirements themselves can be used to motivate the design of an EUSC application, given that they map out a broad range of features that an EUSC application should have. Some of the requirements suggest areas within the domain that require future work before they could be used in the design of an EUSC application. The whole set of requirements is listed in Appendix C.

CHAPTER 4
A DESIGN SPACE FOR EUSC APPLICATIONS

4.1 Chapter Introduction

In Chapter 2 we introduced design spaces, discussed how they can be used in software engineering, and provided some context for how they can be used from prior design literature. In particular, we focused how design spaces have been used in the EUSC domain. Little guidance is given as to how software engineers can create design spaces, either in the literature that discusses design spaces generally, or the design spaces that have been created in EUSC. In Chapter 3, we focused on the first stage of the software engineering process: requirements engineering. [Geyer, 2000] suggests that a link exists between requirements and design spaces, and as such the results of our requirements gathering study are used in the creation of our design spaces for EUSC applications.

One of the main research goals of our work is to create a design space for EUSC applications and evaluate how it can be used to support design in software engineering. We have not been able to identify a suitable design space creation method from prior work and as such have had to specify our own method to create a design space for EUSC applications. Whilst the design space created in this chapter is itself only useful in the domain of EUSC, the method by which it was created is generalisable and could be used in other domains, and hence allow other software engineers to create design spaces for those domains.

The terminology used in design space literature can be unclear, particularly where authors refer to a metaphorical 'design space for X (an artefact)' without giving any definition as to what this design space is. In Section 2.9 we discussed a number of suggested definitions for design spaces [Lane, 1990, MacLean et al., 1991, Baum et al., 2000, Gooch, 2013], and found that whilst there was some agreement, we could draw no definitive conclusions about what design spaces are, how they can be used, and where they fit in the software engineering process. Before describing our design space creation method, we seek to extend the definitions and theory of design spaces to be able to correctly scope for our design space

work.

Design spaces are particularly useful in domains related to ill-structured problems like EUSC because there are a very wide range of ways for applications to support EUSC, and it is not clear which approaches are best, or even what approaches are available. We believe that the structure and rigour that is introduced when creating a design space is invaluable when designers are trying to solve problems such as this. Since we cannot present the whole of our design space in this chapter due to its size, we instead provide an overview of the four main categories of the design space whilst presenting representative images that show small sections of the design space. The design space created in this chapter can be found in Appendix D.

Our review of design spaces in the EUSC domain in Chapter 2 shows in general that there is poor reporting of the methods that were used to create the design spaces. There are an intriguingly large number of design spaces that are based in this domain which were created independently of one another. Whilst these design spaces provide insight into specific areas of SC and mashups, they are often quite limited in scope. Thus, we do not feel that any of the design spaces present an adequate overview of the domain, and that a more comprehensive approach is required. In Section 4.3, we describe a method that we use to create a design space for EUSC that can be extended to other domains with minimal adaptation.

Since the method described in this chapter has not been used before, we analyse the stages that make up the method to identify their relative importance in our case, and discuss how the stages might operate if the design space were being created for a different domain. Two of the stages of the method incorporate work that is described in other chapters of this thesis: the requirements gathering study we described in Chapter 3, and the results of a study we describe present in Chapter 6 to evaluate how design spaces can be used to generate new designs by novice designers.

The aim of this chapter is to report the creation of a design space for EUSC applications. We have identified that the vocabulary used in design space work does not allow us to adequately identify what it is we will be creating, and as such we believe that this vocabulary needs to be refined. We also recognised that there are no documented well established design space creation method. Thus, our aim is broken down into the following concrete objectives:

1. Provide a set of definitions for design spaces that provide a better vocabulary for the design space domain.
2. Describe a generalisable method for creating a design space.
3. Demonstrate the design space creation method by creating a design space for EUSC applications.
4. Provide an overview of our design space for EUSC applications.

4.2 Design Space Theory

In Section 2.9, we discussed prior literature that attempts to define what design spaces are, what they can be used to describe and what they can contain. We found that there were a number of inconsistencies in the definitions of design spaces as presented in the literature, and little consensus in terms of the terminology that is used in practical implementations of design spaces (e.g. the EUSC design spaces). However, generally the literature suggests that design spaces are a multi-dimensional space containing linked design decisions and related solutions to these decisions. We suggest that design spaces can be much more useful to designers if they are described in a more consistent way. Additionally, we suggest that a more consistent definition of design spaces is important for clarifying and extending the terminology used to describe design spaces to make them a viable design aid.

It is clear that there is a terminology problem with terminology in design spaces – we identified 11 prior design spaces created in the EUSC domain, yet none of them uses this terminology. Terms used include 'design space', 'evaluation framework', 'analytical framework', and 'categorisation model'. All of these terms imply some classification or choice, and we believe they all fall into our definition of a design space.

We acknowledge that it can be useful for designers to make generic references to a metaphorical 'Design Space', for instance to provide authors with a generic throwaway term when they are referring to the design choices that a designer has made, or that they could have made. However, we feel that since some researchers are making use of concrete artefacts that share many features with Lane's original design space [Lane, 1990], that a better set of definitions are required to enable the distinction between these two ideas.

[Lane, 1990] provided the original definition for a design space, which provides a high-level overview of the important properties of a design space:

> *"A multi-dimensional design space that classifies system architectures. Each dimension of a design space describes variation in one system characteristic or design choice. Values along a dimension correspond to alternative requirements or design choices."* – [Lane, 1990]

[Gooch, 2013] suggests three groups into which design spaces can be classified based upon their contents and how they are used: conceptual design spaces, generative design spaces, and evaluative design spaces. We do not believe that design spaces should be classified based on their contents as this provides unnecessary restrictions on designers as to what they can include in a design space they they create. For instance, it might be useful for them to consider abstract topics in the domain (e.g. [Mehandjiev and De Angeli, 2012]), concrete features that an application might have (e.g. [Kang et al., 1990]), or something somewhere in between these two extremes. Furthermore, we do not agree that a design space should be classified based on how it is used, since an author could use the same design space for a number of different tasks in the software engineering process.

In this section, we define two types of design space, based on the scope of the design space and provide further discussion as to how these design spaces can be used.

4.2.1 Design Space Scope

Our discussion of the scope of the design space is prompted by [Westerlund, 2005]'s definition, where a design space is "all the possible decisions" and the idea of the metaphorical 'design space'. This conflicts with some other definitions, where the design space is referred to as a finite, usable object [Lane, 1990, MacLean et al., 1991]. Clearly these two definitions are not compatible with one another, even though they provide definitions for the same term.

[Westerlund, 2005] goes further to say that design spaces are normally treated as a *concept* rather than something that is concrete and usable due to the large size of such spaces. Taking this definition to the extreme, a design space could contain infinitely many decisions, and hence would be impossible to fully describe. Our discussions of design spaces do not focus on this idea of being infinite, but we feel that it is important that our definitions support (clarify) the terminology that has been used in design spaces thus far.

The examples of EUSC design spaces that we discussed in Section 2.13 (e.g. [Aghaee et al., 2012, Mehandjiev and De Angeli, 2012]) clearly do not fit into this category of theoretically infinite spaces, and instead are concrete, and ultimately useful for a design task: either evaluation of the design of current applications, or to aid in the generation of new designs.

We believe that it is important to make the distinction between this theoretically infinite, and hence unusable, space and the concrete object that can be used in the design process. Based on the above two views of design spaces, we suggest two definitions of design space related to the abstractness and scope of design spaces: *Implicit Design Spaces* and *Explicit Design Spaces*.

> **Definition 12: Implicit Design Space.**
> An Implicit Design Space is the theoretically infinite space containing *all* possible design decisions for a particular design problem. Since it is theoretically infinite in size, the implicit space cannot be fully described [Westerlund, 2005]. References to an implicit design space would normally be of the form 'the design space for X', referring to the metaphorical space in which all the design choices that could be made in X exist.

> **Definition 13: Explicit Design Space.**
> An Explicit Design Space is a concrete, finite model of the implicit design space, restricting decisions based on some criteria. Aspects of design solutions can be modelled in an explicit design space. Since we cannot describe the implicit space, creating a concrete and explicit model is a necessary step in describing design spaces and making them usable.

By introducing the notion of abstractness to the vocabulary of design spaces, we allow both the metaphorical 'design space' that authors often refer to when discussing the design of artefacts, as well as the idea that a design space can be a concrete object that can be used to complete various tasks in the design process. The rest of this section discusses further classifications of explicit design spaces, based on how designers can use them. By definition, implicit design spaces cannot be classified in this way.

4.2.2 Design Space Usage

Design space usage considers how a designer can use an explicit design space in the design artefact. [Gooch, 2013] suggests two such uses: evaluating existing designs, and generating new designs. For either of these two uses of design spaces to be viable, a designer using a design space needs to be able to ascribe some 'value' to each decision or combination of decisions and hence be able to say whether they are 'good' or 'bad' in a given context. For example, a client-server architecture makes more sense for a web-based application than for an offline desktop app, and as such the value associated with the design choices might change when certain combinations of design choice are considered.

Given some value judgement associated with the design decisions and solutions in the explicit design space, [Gooch, 2013] suggests that there are two potential uses based on two different theories of design: generation of new designs (based on prescriptive design theory) and evaluation of existing designs (based on explanatory design theory). Jul [Jul, 2002] describes explanatory and prescriptive design theory. Explanatory theory identifies existing design features: *"Y* is *thus because of X"* [Jul, 2002]; prescriptive design theory on the other hand describes switches the focus to design features in *"Because of Y, X* should be *thus"* [Jul, 2002].

We do not believe that it is practical to define a design space based on how it is used, given that the same design space artefact could be used to perform several of the tasks that we have identified for which they can be useful. That is, given an explicit design space and some idea of value for each of the design choices in the explicit space, a designer could use it for either evaluation of designs or creation of new designs.

The most obvious use for design spaces in design generation is for the designer to choose elements from the space to add to their design. Alternatively, they may simply use the decisions presented in the space to prompt them as to choices that they might want to make. When using explicit design spaces generatively, we might also want to consider design rationale to record the reasons that a designer has for choosing particular design solutions over others. We believe that an explicit design space as an artefact should remain independent of subjective concepts like criteria, and that they have a much greater value when we consider specific designs either being evaluated or generated using the explicit design space, rather than being added to the space itself. That is, the rationale for a particular design choice

might be appropriate if the designer were creating a Web-based EUSC application, but not appropriate if they then designed a mobile version.

In Section 2.9.7, we discussed the tasks associated with the software engineering process, and identified links between design spaces and each of requirements engineering, design, and validation. Using the above definitions and discussions of how design spaces can be used, we are now in a position to suggest how design spaces should be used in software engineering.

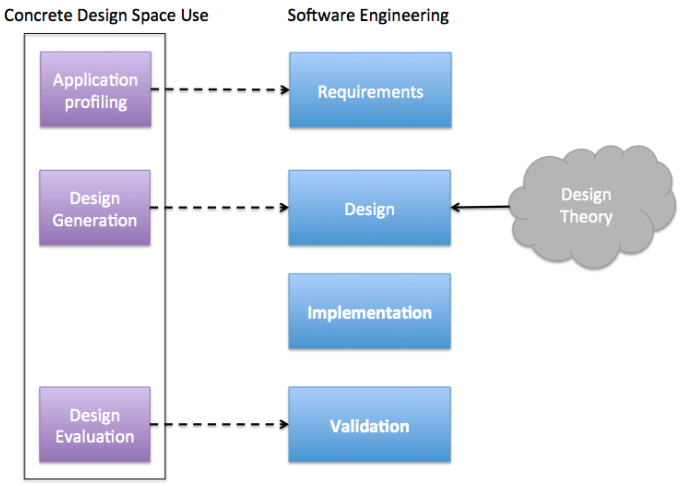


Figure 4-1: The mapping between design spaces and software engineering.

Figure 4-1 shows the stages in software engineering, and how an explicit design space can be used in each of these stages. Specifically, explicit design spaces can be used to profile existing applications to help the generation of requirements, used generatively in the design stage to help create new designs, and used evaluatively in the validation stage to evaluate or validate the design that has been created. In the remainder of this theses, uses of the term 'deign space' refer to explicit design spaces rather than implicit, unless specified otherwise.

4.3 Design Space Creation Method

In this section, we will describe the method that we used to create our explicit design space for EUSC. Once we have created the design space, we will be able to use it for application profiling, design evaluation and design generation. Before specifying the method for creating the design space itself, we will discuss how other design spaces in the domain have been created, and motivate why a well-specified method is needed for creating explicit design spaces.

We first discuss prior approaches to creating design spaces, identifying the methods that were used to create them. We then clarify the structure that underpins the design space, before describing the method that we used to create it. Our method only specifies the creation of the

main elements of the space: design decisions, solutions, and the links between them. Our method does not cover the addition of other attributes such as value or design rationale since they are specific to one instance of the design space being used (e.g. to generate a design for a specific EUSC application), rather than being applicable to the design space as a whole.

One of the main problems with this lack of specification of the methods used to create design spaces is that it makes them a much less attractive proposition for both designers and researchers. We seek to remedy this by specifying our own method that other designers and researchers can use to create explicit design spaces in any domain.

4.3.1 Existing Design Space Creation Methods

In Section 2.13, we discussed a number of design spaces that were created within the EUSC domain (or, more specifically for mashups). The presence of design spaces for EUSC were part of the motivation for us to create our own explicit design space in the domain. A common weakness of these prior design spaces is in the lack of specification of the method that was used to create them.

Amongst these prior design spaces, the most commonly reported method for creating them was to perform a review of applications in the domain that were available for the authors to review. For instance, [Aghaee et al., 2012] reviewed 22 existing artefacts, [Grammel and Storey, 2010] reviewed 6, and [Minhas et al., 2012] reviewed 12. Other approaches include using key concepts from the literature in the domain as dimensions of the design space (e.g. [Mehandjiev and De Angeli, 2012] identified key End-user Development [EUD] concepts and a service-oriented computing life cycle as dimensions when assessing EUD support in SC applications). Another approach used in the prior EUSC design spaces was to create dimensions of the design space that were based on the requirements that they had already identified for the application that they were trying to create [Albinola et al., 2009]. Other design space creation methods were either vague [Na et al., 2010], or not specified at all [Fischer et al., 2009, Pietschmann et al., 2010, Brønsted et al., 2010, Tuchinda et al., 2011].

An application review was the most common approach for creating the EUSC design spaces. Whilst this approach seems to be a perfectly valid one, none of the creators of the design space provide much insight into *how* the application review was carried out. [Grammel and Storey, 2010] give the strongest description of all of the authors who performed a application review: *"qualitative, exploratory tool analysis"*. Some discussion is provided for the applications that were chosen to be reviewed, usually based on the availability of the application in question. For instance, if the reviewed application was available at the time then the author could review the application by using it, otherwise they would need to access secondary materials such as screenshots, videos, etc.

[Albinola et al., 2009]'s approach of using requirements to form a design space reinforces the link between requirements and Design Spaces, which is also noted by Geyer [Geyer,

2000]. This link between requirements and design spaces implies that standard requirements gathering methods can be used to generate requirements that can then be transformed into a design space. This demonstrates that the requirements gathered in Chapter 3 could be incorporated into our explicit design space.

We believe that the deficiencies we have identified here motivate the need for a well-specified, robust and repeatable method for creating a design space. We exemplify our method in the domain of EUSC, but then discuss its generalisability and how it could be applied in other domains.

4.3.2 Design Space Structure

When we consider a practical implementation of an explicit design space, we must consider the model that underpins this implementation. Specifically, we need to identify how to describe the relationships between the decisions and potential solutions. The only candidate model from the EUSC design spaces is used by [Aghaee et al., 2012], which is first suggested by [Nowak and Pautasso, 2011]. This knowledge meta-model is described in Section 2.13.1.

[Nowak and Pautasso, 2011]'s meta-model describes the relations between the design decisions and their associated potential solutions, but does not consider how related design decisions are grouped together. In his original work on design spaces, [Lane, 1990] suggests a split between functional design decisions and structural design decisions, i.e. functions that the application performs, and the architecture of the application.

The functional and non-functional categories describe a broad range of topics related to a piece of software. However, a number of the prior design spaces cover other aspects that do not fall within these categories, and as such we added a non-functional category to the design space. We added a final category to the structure of our design space based on findings part way through carrying out the method. Specifically, we identified that a very large number of design decisions and solutions that related to the services in EUSC. Thus, we added a category that reflected the interactions that the user can have with the entities in composition. Whilst it may seem that this category is specific to the EUSC domain, it would also be applicable to other domains, e.g. contacts in a chat application.

It should be noted that the separation of the space into categories are separated for convenience of representation and for ease of interaction with them. The structural category is a sub-category of the non-functional category, and the entity category contains design decisions that could be classified either within the functional category or the non-functional category.

Our explicit design space has the following properties:

- **Category:** The design space is split into four categories based on the model of Baum & Geyer [Baum et al., 1998, Geyer, 2000] and the additions specified above. The four

categories in our design space are:
  - *Functional*: Functional properties of the SC application – the things it should **do** (not including direct interactions with entities).
  - *Non-functional*: Non-functional properties of the SC application (not including the structure or architecture of the application).
  - *Structural*: Properties that relate to the structural or architectural design of the SC application.
  - *Entity*: Interactions with and the representation of entities in the domain.
- **Decisions** – The design decisions or issues that needs to be addressed.
- **Potential solutions** – Various options or potential solutions to each design decision.

In an explicit design space, we believe that it is more important that similar design choices are 'near' one another within the space than enforcing strict rules upon the category in which design choices must reside. This is because the design space is split into categories for convenience of representation – representing it as a single monolithic space could be overwhelming to any designers (particularly novices) who decide to use it for some design task. Placing similar concepts near to one another means that the designers using the design space for profiling or design generation can see similar decisions at a glance, rather than having to navigate around the space for design choices that might be associated with the one that they have just made or identified.

4.3.3 Method

In this section, we present our method for creating an explicit design space. The resources that we had before creating the design space were: several EUSC design spaces, EUSC literature, EUSC applications, requirements for EUSC applications (reported in Chapter 3), and the results of our study exploring design spaces in design generation (reported in Chapter 6). Taking these available resources into account, we outlined four main stages of a method to create a design space. It is worth noting that in our implementation of this method, the base design space is creating during stage 1, and the other stages augment and adapt this design space. The four stages of our initial design space creation method were:

1. **Explicit Design Space Collation:** The contents of design spaces already identified in the domain were collected together. Where overlapping concepts were found, they were consolidated.
2. **SC Literature review:** Domain literature was reviewed to identify design decisions or solutions for EUSC applications that were not already present from the design spaces in stage 1.
3. **EUSC Application review:** Applications in the domain were reviewed to identify the following:
   - Find solutions for decisions already in the design space
   - Find decisions for solutions already in the design space

- Consolidation: break apart complex or compound decisions or solutions into their constituent parts.
- Iteration of the application review until the size and structure of the design space does not change.

4. **Requirements Integration:** Augment the design space with the results of the requirements gathering study (presented in Chapter 3).

After creating an initial version of the design space by completing these four stages, we were performed a study to evaluate how design spaces could be used in the design process. The results of the study were also integrated into the design space.

5. **Design space study results integration:** Augment the design space with the results of the design space study (presented in Chapter 6).
6. **EUSC Application review II:** We repeated the application review a second time so that the design space reflects new applications that are available.

In the remainder of this section, we will describe what each of these stages entailed, and how the design space was changed as a result. Our description of the stages that draw on work from other chapters of the thesis will provide a brief overview of the aims of that work, before discussing the effects on the design space. The requirements gathering study is described in detail in Chapter 3, and the design space study is described in detail in Chapter 6.

Throughout the section, we will present a series of tables that show how the size and structure of the explicit design space changes after each stage. We will try to compare the effects of each stage once the whole method has been described. We use this as a motivation for discussion rather than using it to come to any concrete conclusions, given that the size and structure of the design space does not describe its contents. However, size and structure are the only quantifiable property of the explicit design space that we can assess.

Stage 1: Explicit Design Space Collation

The first stage in our explicit design space creation method creates an initial design space based upon an amalgamation of the design spaces that we described in the last section. To make this process as easy as possible, we chose to start off with one of the EUSC design spaces as an initial base, and then integrated the design decisions and solutions of the other design spaces into the facets described in the first. Given our choice to use [Nowak and Pautasso, 2011]'s knowledge meta-model that describes [Aghaee et al., 2012]'s design space, it seemed like the natural choice to use as an initial design space for this step.

The only modification that was required to [Aghaee et al., 2012]'s design space was to categorise the design decisions within the categories identified earlier: functional, non-functional, and structural. The entity category had not been identified at this stage and as

such is not included.

After categorising [Aghaee et al., 2012]'s design space, we reviewed each of the other explicit EUSC design spaces to identify potential design decisions and solutions, and then find the right position in our design space at which to incorporate the identified decisions or solutions. The prior explicit design spaces were reviewed in the following order:

1. [Grammel and Storey, 2010]
2. [Minhas et al., 2012]
3. [Na et al., 2010]
4. [Mehandjiev and De Angeli, 2012]
5. [Albinola et al., 2009]
6. [Fischer et al., 2009]
7. [Pietschmann et al., 2010]
8. [Brønsted et al., 2010]
9. [Tuchinda et al., 2011]

As the new design elements (decisions or solutions) were augmented with the old, a number of different conditions arose that needed to be addressed:

- The new decision or solution is not in the old design space. The new decision or solutions can be simply inserted into the relevant position.
- The new decision or solution being added is already in the design space. The new decision or solution is already in the design space, so nothing needs to be done.
- The new solution is not in the design space, but its parent decision is. The new solution can be simply added as a solution to the design decision already present in the design space.
- The new solution is not present in the design space, but a more open-ended version of that solution is present in the space already. The old already-present solution is transformed into a decision, so the solution can then be added as a solution to that decision.

To see the influence that each of the EUSC design spaces had on our design space, we identified the source of each of the design elements. The number of elements that were added from each source is shown in Table 4.1. This information is shown graphically in Figure 4-2. Table 4.1 shows all the design elements that were identified from each source, meaning that there is some overlap between the design elements. Thus, the analysis values presented in Table 4.1 do not align exactly with the values presented in the latter stages of the method.

We can see from Table 4.1 and Figure 4-2 that the different sources had varying effects on the size and structure of the design space as a whole. Some of the prior design spaces focused on the functions the EUSC application would complete, others focused more on non-functional aspects such as user interaction, whilst others looked more into the architecture of the application. Few considered all three in any detail, showing that even when considering the same domain, different methods for creating design spaces can yield vastly different results. The contents of the design space that was created also depended on the focus of the paper in which they are described, for instance several authors focused on End-user

Table 4.1: Number of new design properties from each prior explicit EUSC design space, grouped by category.

Source	Functional		Non-Functional		Structural	
	Decision	Solution	Decision	Solution	Decision	Solution
Aghaee	2	5	6	14	2	6
Grammel	4	17	3	17	0	2
Minhas	0	6	4	14	4	9
Mehandjiev	9	14	0	0	0	0
Albinola	2	1	1	3	0	0
Na	2	4	2	11	1	13
Fischer	0	4	0	5	1	3
Pietschmann	2	8	0	3	6	28
Brønsted	5	14	0	0	5	13
Tuchinda	1	2	0	0	3	17

development and mashups, and the design spaces in these works focused on non-functional aspects linked with end-users, such as representation and how the user can interact with such an application.

After grouping together the contents of the prior design spaces, it was clear that there was some duplication of elements, and design solutions that may need to be split up to be at the same level of granularity as other solutions within the design space. To address this, we performed another task that we called "reorganisation and consolidation". As part of this task, we removed any duplicates from the design space, and moved concepts that were misplaced. In order to make sure the design space remained consistent after each stage of our method, we performed this reorganisation and consolidation activity after each stage.

Since a number of the design spaces upon which ours is based focus on mashups rather than EUSC, the terminology within our design space was mashup-specific in some cases. Where applicable, we generalised mashup-specific concepts to their SC or EUSC counterparts. For instance, where the initial explicit design space referred to "process mashups" or "data mashups", the updated design space instead referred "process composition" and "data composition", respectively. Since the terminology is contained within each of the decisions or solutions, the structure of the design space did not change in this generalisation stage. It should also be noted that the renaming process did not change the meaning of any of the design elements.

Stage 2: General SC Literature Review

The second stage in our explicit design space creation method was a review of the literature in the domains of SC and EUSC, with the exception of work already presented in the DS collation stage. The contents of this stage have considerable overlap with the first, in that a large number of the concepts identified in the literature were also present within the explicit

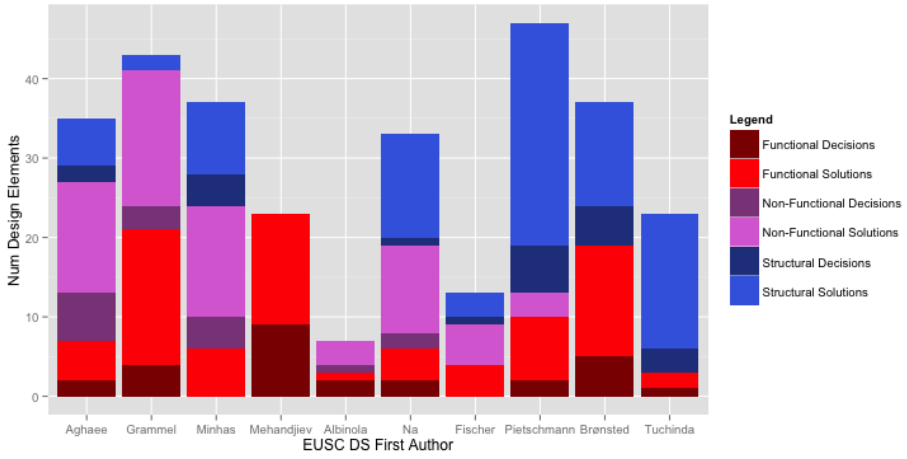


Figure 4-2: Number of design elements identified from each of the sources of the design space collation stage.

EUSC design spaces used in the first stage. We elected to perform this stage after the design space collation stage because of the structure that is inherent in the explicit design spaces that could be used as scaffolding for our design space, which is not present in the SC literature.

We can see from the topics we discussed in Chapter 2 that there are areas of the design space that do not adequately represent the research in the domain. Decisions and solutions added at this stage covered topics such as the user of the application, (classified based on their domain knowledge and technical knowledge [da Silva et al., 2010, Nestler et al., 2011]), the abstraction level at which the application operates [Cappiello et al., 2011a], and the domain in which the application allows composition [Cappiello et al., 2011a]. The structure of the functional category was also modified based on the dynamic SC life cycle work by [da Silva et al., 2008]. Finally, work on technologies supporting SC was incorporated from standards documents to provide additional information about how SC can be achieved technically [W3C, 2004a, W3C, 2005b, W3C, 2010, W3C, 2007, W3C, 2005a, Kopecky et al., 2009].

After the literature review had been completed and the design space had been extended, we performed another reorganisation and consolidation stage to ensure that the design space remained consistent and the terminology remained correct. The changes in the structure and size of the design space following the literature review are shown in Table 4.2.

The literature review resulted in relatively few additions to the design space compared to the more specialised DS collation stage, but as we have discussed there was a large amount of overlap between the concepts identified in the general literature review and the DS collation stage.

Table 4.2: Changes in structure and size of each design space category after the literature review.

	Functional		**Non-Functional**		**Structural**		**Entity**	
	Decision	Solution	Decision	Solution	Decision	Solution	Decision	Solution
Initial	26	64	19	47	26	78	0	0
Removed	0	0	0	6	1	2	0	0
Added	4	0	4	0	8	12	0	0
Modified out	5	1	1	1	3	0	0	0
Modified in	5	0	1	2	3	0	0	0
Final	**30**	**63**	**23**	**42**	**33**	**88**	**0**	**0**
Change	+4	-1	+4	-5	+7	+10	0	0

Stage 3: EUSC Application Review

The next stage of our method was to review existing SC applications to identify design decisions and solutions. A number of the prior explicit EUSC design spaces identified this as the process by which the design space was created, but none provide any insight into how this review is performed, meaning that we had to devise our own method.

[MacLean et al., 1991] suggest a number of heuristics for performing analysis of a design space, which can be considered either locally or globally. Local heuristics relate to single design decisions and solutions, and the relations with their respective solutions and decisions. Global heuristics relate to the design space as a whole. In their original work, [MacLean et al., 1991] refer to 'Questions' and 'Options', but we remain consistent with our previous use of terminology: decisions and solutions. A selection of local heuristics were useful to us in this stage of our method:

H1. **Use Decisions to generate Solutions.**
Apply identified questions to the application in order to identify how they solve that design decision [MacLean et al., 1991].

H2. **Use Solutions to generate Decisions**
Progress through already identified solutions and identify the Decision that each solution is trying to solve [MacLean et al., 1991].

H3. **Consider distinctive Solutions**
Extreme solutions can provide interesting additions to the design space, and presenting aspects that may lead to trade-offs varying solutions.

We devised an iterative method for performing a application review based on the group of local heuristics listed above. We applied these heuristics to each of the applications and based on the contents of our explicit design space at the beginning of the stage. Before describing our application review method, we will first discuss the applications to be reviewed.

Chosen EUSC Applications Several of the prior explicit design spaces used to motivate our own were built on application reviews, and even those that were not built on application

Table 4.3: Selected applications for the first application review.

Application	Context	Platform	Domain
Atooma	Mobile	Android	Mobile, Social
AutomateIt	Mobile	Android	Mobile
Tasker	Mobile	Android	Mobile
IFTTT	Web	N/A	Social, Productivity
OnX	Web (Composition), Mobile (Execution)	N/A, Android	Mobile
Yahoo! Pipes	Web	N/A	RSS
Zapier	Web	N/A	Enterprise
Automator	Desktop	OSX	Desktop, Productivity
Quartz Composer	Desktop	OSX	Multimedia

reviews profiled available applications in order to categorise the design decisions made within them. This provides us with a long list of potential applications that could be used for our application review.

For the majority of the prior works containing EUSC design space, the applications that were used in the creation methods no longer exist. This means that to incorporate them into our application review, we would have to review secondary resources of the designs of these applications such as research papers, screenshots and videos. It is unlikely that we would be able to glean any more insight than is provided in the prior EUSC design spaces in which these defunct applications were originally reviewed.

We felt that is was particularly important for our application review to focus on existing applications given that the review of these applications could be carried out in a more comprehensive and systematic way than using secondary sources. The information that can be gathered from using the applications themselves dwarfs the amount that can be gathered by using a secondary source such as documentation or videos[17].

Of all of the applications reviewed or profiled in the EUSC design spaces, only one remains available: Yahoo! Pipes. Yahoo! Pipes was reviewed by so many of the creators of design spaces that it was cited as being a standard "benchmark" against which new mashup development applications are compared [Mehandjiev and De Angeli, 2012]. The applications that we selected for this application review are listed in Table 4.3. All EUSC applications that are discussed in this section are described in more detail in Section 2.7.

Another set of applications was reviewed in a secondary application review after the first version of our design space had been finalised, to reflect EUSC applications that were released in the time after this first version had been finalised. These applications are listed in Table 4.7.

[17] `http://www.lib.vt.edu/help/research/primary-secondary-tertiary.html`

Application Review Method The method used in our application review was broken down into five stages, based on the heuristics presented by [MacLean et al., 1991]. It was designed to be systematic, and operates in an iterative manner. Applying the application review method iteratively means that we can alternately assess decisions and solutions to generate better solutions, which agrees with work by [Schön, 1987]. The stages of our application review were:

1. **Identify solutions.** For each decision in the explicit design space, identify how the tool solves that decision. If that solution isn't in the design space, add it.
2. **Identify solutions.** Identify solutions in the application that do not address any decision within our space, and add decisions to the space accordingly.
3. **Identify decisions.** For each solution in the design space, identify it within the application (if it exists), and then identify what the application is trying to solve with that particular solution. If the decision that is being solved is not present, add it.
4. **Solution breakdown.** For each solution in the design space, determine whether it can be broken down into more fine-grained solutions. If it can, transform the solution into a decision and add the fine-grained solutions as solutions.
5. **Reorganisation & Consolidation.**

We performed the first part of our application review in May 2013 on each of the applications in the first set of applications identified above. We performed this stage iteratively until no new design elements were added to the design space, which occurred on the third iteration. The changes in the size and structure of the design space after the first two iterations of the application review are shown in Table 4.4.

Table 4.4: Changes in structure and size of each design space category after the two iterations of the first application review.

Application Review: Iteration 1								
	Functional		**Non-Functional**		**Structural**		**Entity**	
	Decision	Solution	Decision	Solution	Decision	Solution	Decision	Solution
Initial	30	63	23	42	33	88	0	0
Removed	0	2	2	0	1	3	0	0
Added	1	15	8	29	3	16	7	54
Modified out	2	1	3	4	0	0	0	0
Modified in	2	0	3	2	0	0	1	3
Final	**31**	**75**	**29**	**69**	**35**	**101**	**8**	**57**
Application Review: Iteration 2								
	Functional		**Non-Functional**		**Structural**		**Entity**	
	Decision	Solution	Decision	Solution	Decision	Solution	Decision	Solution
Initial	31	75	29	69	35	101	8	57
Removed	0	0	0	0	0	0	0	0
Added	6	15	0	5	3	3	0	1
Modified out	0	0	0	0	0	0	0	0
Modified in	0	0	0	0	0	0	0	0
Final	**37**	**90**	**29**	**74**	**38**	**104**	**8**	**58**
Change (Iteration 1)	+1	+12	+6	+27	+2	+13	+8	+ 57
Change (Iteration 2)	+6	+15	0	+5	+3	+3	0	+1
Change (overall)	**+7**	**+27**	**+6**	**+32**	**+5**	**+16**	**+8**	**+58**

The main change to the design space after the application review was the identification of the entity category, with a large number of solutions being added into this category of the design space. The application review had a similar level of effect to the initial DS collation, and a much greater effect than the literature review.

Stage 4: Requirements Integration

Design spaces are suggested as an effective way of representing sets of requirements [Geyer, 2000], so it makes sense for us to incorporate any requirements we were able to gather that are not already present in the explicit design space. Chapter 3 describes a tailored requirements gathering approach based on providing end-users with scenarios, a prototypical demonstrator and other examples of designs for SC applications. The method used to gather the requirements is described in detail in that chapter, and we present the requirements gathered in this study in Appendix C.

To augment our design space with the requirements we gathered, we first re-organised the set of requirements to use the same basic structure as our design space: hierarchically organised within the 4 categories of functional, non-functional, structural and entities.

In the requirements elicitation process, our requirements were elicited from codes that were derived from that data gathered in the study. These codes were organised into categories. The similarity in structure between the requirement codes and the design space meant that it was a relatively simple process to systematically analyse them and add them to the design space. To do this we identified the code that was associated with each of the requirements, and then looked through the design space for a decision or solution with which the code could be associated. If the identified element in the design space was a decision, then the code was simply added as either a solution or a sub-decision. However, if the identified element in the design space was a solution itself, then was transformed into a decision before a solution or sub-decision could be added from the code.

As with each of the other stages, once the requirements had been integrated into the design space, we performed another round of reorganisation and consolidation to ensure the design space remained consistent. The changes to the size and structure of the explicit design space are shown in Table 4.5.

The requirements gathering phase had a lesser effect than either the DS collation stage or the application review. However, given that the size of the explicit design space increases with each stage, it is to be expected that the effect of each stage would diminish as more stages are carried out.

Table 4.5: Changes in structure and size of each design space category after the inclusion of data from requirements.

	Functional		**Non-Functional**		**Structural**		**Entity**	
	Decision	Solution	Decision	Solution	Decision	Solution	Decision	Solution
Initial	37	90	29	74	38	104	8	58
Removed	0	0	1	2	2	0	1	0
Added	5	16	4	10	5	8	7	24
Modified out	0	2	0	3	0	0	0	0
Modified in	0	4	0	0	0	0	1	2
Final	**42**	**108**	**32**	**79**	**41**	**112**	**15**	**84**
Change	+5	+18	+3	+5	+3	+8	+7	+26

Stage 5: Integration Design Space Study Results

The last new stage of our method is the integration of the results of our design space study, which we discuss in Chapter 6. It is important to consider these changes here, because we can see that the use of the explicit design space as a generative design aid can also facilitate new knowledge being added to the design space.

In Chapter 5, we present a tool that can be used by designers to generate new designs in the domain in which a design space has been created. Later, Chapter 6 presents a study where participants used this design tool, containing our explicit EUSC design space, to generate new designs for EUSC applications. The aim of the study was to determine how the addition of a design space can assist users with the in the creation of designs, and to investigate the effect that this assistance can have on the designs that are produced. The method for this study is described in Chapter 6. In the study, participants were invited to enter their own design ideas in addition to those found within the design space. A number of the custom design solutions presented by the participants in the study were identified as being historically-novel in the domain. Following the collation of the results from the study, we added these new, novel design elements to the design space.

The changes to the size and structure of the design space after the changes were added from the design study are shown in Table 4.6.

Table 4.6: Changes in structure and size of each design space category after the inclusion of data from the design study.

	Functional		**Non-Functional**		**Structural**		**Entity**	
	Decision	Solution	Decision	Solution	Decision	Solution	Decision	Solution
Initial	42	108	32	79	41	112	15	84
Removed	0	0	0	0	0	0	0	0
Added	7	26	5	10	1	8	0	3
Modified out	0	0	0	0	0	0	0	0
Modified in	0	0	0	0	0	0	0	0
Final	**49**	**134**	**37**	**89**	**42**	**120**	**15**	**87**
Change	+7	+26	+5	+10	+1	+18	0	+3

Table 4.7: Selected applications for the second application review.

Application	Location	Context	Platform	Domain
Condi	https://play.google.com/stor	Mobile	Android	Mobile automation
E-Robot	https://play.google.com/stor	Mobile	Android	Mobile automation
Automated Device	https://play.google.com/stor	Mobile	Android	Mobile automation
IFTTT (iOS)	https://itunes.apple.com/gb/	Mobile	iOS	Mobile, social, productivity
IFTTT (Android)	https://play.google.com/stor	Mobile	Android	Mobile, social, productivity

The integration of the novel design choices generated by our participants in the design study had a similar level of effect to the requirements gathering study, although more design elements were identified in the functional and structural categories, with very little being added to the entity category.

EUSC Application Review II

EUSC applications are released relatively frequently, particularly on the Android platform. To account for newly released applications and updates to the applications in the first application review, we decided to perform a second application review.

Since our application review method utilises each of the design elements in the design space to motivate the identification of new elements, the more elements in the design space, the more considerations can be made for each application that is being reviewed. Hence, we decided that a second application review would be the final stage of the design space creation method to maximise its usefulness. We also re-reviewed those applications chosen for the first application review.

We performed the second part of our application review in March 2014 on each of the applications identified in Tables 4.3 and 4.7. We carried out the same iterative method described in the first application review, and continued until no more changes were made to the design space after an iteration. This occurred on the second iteration. The changes in the size and structure of the design space after the first iteration of the second application review are shown in 4.8.

4.4 An Explicit Design Space for EUSC application

In this section, we describe the explicit design space that we created, which gives a broad overview of the design elements that are – or could be – design choices made in an EUSC application. We describe the contents of the explicit design space across four categories that were identified in Section 4.3.2 to break down the whole design space into more manageable sub-sections: Functional, Non-Functional, Structural, and Entity.

Table 4.8: Changes in structure and size of each design space category after the second application review.

	Functional		**Non-Functional**		**Structural**		**Entity**	
	Decision	Solution	Decision	Solution	Decision	Solution	Decision	Solution
Initial	49	134	37	89	42	120	15	87
Removed	0	0	0	0	0	0	0	0
Added	7	26	0	4	0	0	0	2
Modified out	0	0	0	0	0	0	0	0
Modified in	0	0	0	0	0	0	0	0
Final	**56**	**160**	**37**	**93**	**42**	**120**	**15**	**89**
Change	+7	+26	0	+4	0	0	0	+2

Due to the size of the design space, we describe each of the individual design elements in Appendix D. Each element has the following attributes: name, description, element type, stage added, and parent element. Since we cannot describe each element here, we will instead present an overview of each category.

The descriptions of the design elements within our design space for EUSC applications are found in the following sections of Appendix D: Functional category – Appendix D.1 (page 281); Non-Functional category – Appendix D.2 (page 299); Structural category – Appendix D.3 (page 309); Entity category – Appendix D.4 (page 321).

As a clarifying example of some of the contents of the design space, consider the following design elements from the functional category:

- **Composition type**: [Decision] – DS collation stage
  The "type" of composition that the tool supports, e.g. process, data, or interface.
  Solves: Construct composition
- **Logic/process composition**: [solution] – DS collation stage
  The composition involves connecting up a series of processes.
  Solves: Composition type
- **Presentation/UI composition**: [solution] – DS collation stage
  The tool allows the user to compose new user interfaces for composites.
  Solves: Composition type
- **Data composition**: [solution] – DS collation stage
  The composition process involves modifying a set of data by composing services together.
  Solves: Composition type

A set of design elements from the same region of the design space are shown in Figure 4-3. The remainder of this section provides an overview of each of the four categories of the design space, describing the major groups of design decisions that are present in each category.

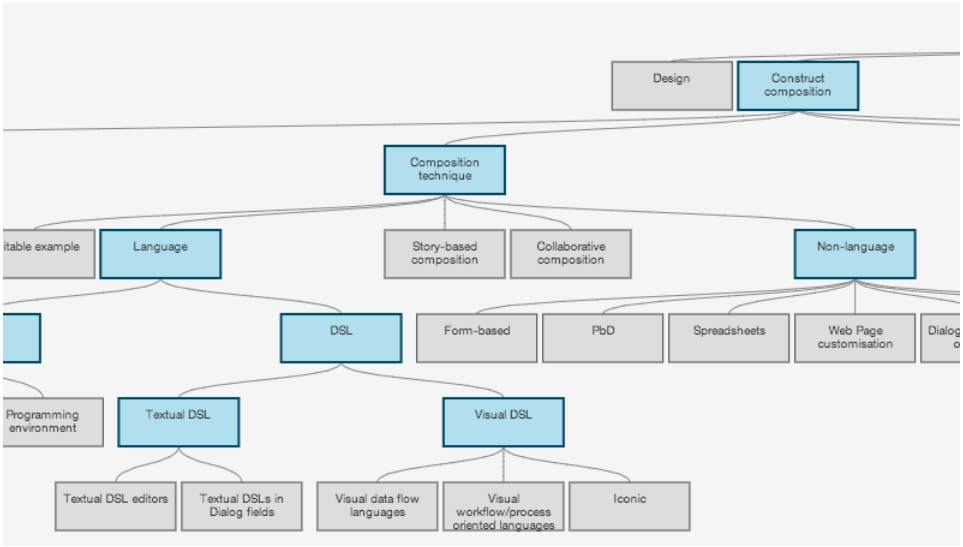


Figure 4-3: A set of example design elements from the functional category of our EUSC design space.

4.4.1 Functional Category

The Functional category is the section of our design space that describes the functions that an EUSC application can complete (this is similar to a collection of functional requirements for a software artefact). As is discussed in Section 4.3.2, we feel that it is important to ensure that similar concepts are near one another rather than strictly adhering to the categorisation mechanism for every single design element. For instance, when considering the discovery process for components in the composition process, it is more useful to consider how the application might support discovery at this stage, rather than moving sub-domains relating to discovery into the non-functional part of the design space. Thus, our functional space contains the main functions that an EUSC application could support, along with design choices that might not be considered functional on their own, but are clearly very closely related to a functional design choice.

The largest group of design elements in the Functional category are based on the stages of the EUSC life cycle, first identified by [da Silva et al., 2008], and later expanded by [Mehandjiev and De Angeli, 2012]. Our design space contains an amalgamation of these two life cycles, where each of the stages of the life cycles are augmented with work from other authors (such as the pre-existing design spaces for EUSC applications or Mashup Development Environments [Aghaee et al., 2012, Minhas et al., 2012, Na et al., 2010, Grammel and Storey, 2010]).

The stages of the life cycle that are present in the functional category cover those that are explicitly supported in current examples of EUSC applications, those that are only applicable in automated composition (e.g. specification of the required composite) [da Silva et al.,

2008], and those that are currently identified as being external to the EUSC application (e.g. inception or domain analysis [Mehandjiev and De Angeli, 2012]). We felt that it was important to include all of these stages since they have been identified in prior literature as aspects that may benefit from tool support, and it is possible that they are not present in the design of current EUSC applications because designers of these applications have not previously thought to support these tasks.

After the implicit stages comes the realisation of the composition process. Users need to discover the components that they need to use to create their composite, construct the composite through some coordination of the discovered components, and then ensure that it (a) works, and (b) does what they expected it to. Each of these stages is supported in current EUSC applications through various mechanisms (e.g. discovery might be supported with browsing, searching, etc.), although there is no particular approach that is universally favoured in any case.

The later stages of the life cycle are also represented, split into publication (submitting the created composite to some repository where it could be discovered by others), management (editing the composite based on changes in the user's requirements), and execution.

EUSC applications also provide other functionality on top of explicitly supporting the EUSC life cycle, which are also presented in the functional category of the explicit design space: component management, user management, device management, and global settings. Component management covers aspects such as the addition of new components to the application, which is not technically part of the EUSC life cycle as it would not normally be the end user who would do this. However, they might send requests for particular components to the application developer, or some such. User management revolves around the application supporting user registration where the user can create a profile and associate their created composites with it. Features associated with user management include profile management, social networking, billing, etc. Device management is applicable to EUSC applications that are available across multiple platforms or devices (this is also dependent on user profiles). For instance a user might log in to the EUSC application on their tablet and their phone, and have a number of composites that are only needed on the tablet, whereas others are needed on the phone. Global application settings are settings that apply across all of the services within the application, such as units, location granularity, sensor reading frequency, etc.

4.4.2 Non-Functional Category

The non-functional category is a very broad category, and contains non-functional aspects of EUSC applications that have not been relocated to related topics in one of the other design space categories. The non-functional category contains a number of separate, high-level design decisions: representation, online communities, learning support, and aspects relating to the target user or domain.

The first major group of design decisions in the non-functional category is representation, which is split into the representation of the composition process, and of the components contained within it. Note that the representation of components could be considered whether they are being represented within the composition stage, or without. Within composition representation we also have the explicit or implicit representation of control and data flow in the composition process.

The second major group of design decisions in the non-functional category is the relation to the online community, which encompasses how the application supports collaboration between users, what can be shared between them, and how the online community succeeds in sharing these entities.

The third major group of design decisions relates to the usability of the application. This encompasses two main areas: learning support and error recognition. Learning support presents mechanisms for how the user can learn how to use the application, and get support as and when they require it. It also discusses the learning curve associated with using the application. Error recognition discusses how the application indicates to the user that something has gone wrong, or might have gone wrong in the composition process, so that they can work out how best to deal with it.

The final group of non-functional design decisions related to the target user, who is classified based on their skill and the context in which they would use the application. The target domain is simply the domain in which the components within the EUSC application operate, for instance they may all be mobile services, or they may also encompass social networking services.

4.4.3 Structural Category

The structural category (originally suggested by [Lane, 1990]) is a subset of the non-functional category that focuses on design choices relating to the architecture or structure of the application, and in our case, the structure of the composition itself. The structural category is split into four sections: the possible structure/architecture of composition, the structure/architecture of the application, the technologies used to support composition, and the 'types' of component that the application supports.

The structure of composition is split into a number of sections. First we have the level of automation that the application supports (full automation, semi-automation, no automation), and the layer at which composition operates (presentation layer, application layer, and service layer). Composition structure also contains liveness, which is the relationship between the flow diagram (or similar) and the composite that it represents. Composition logic relates to the logical operators and similar programming structures that the application supports, for instance loops or conditional statements. Lastly, layout logic relates to the overall layout of the composition section of the application (text-based or canvas-based, etc.) and the use of

templates within the application.

The application's structure identifies what the application *is*, in terms of the type(s) of application that the user can use to perform composition and still be considered as using the application (e.g. IFTTT was a Web page and later both mobile apps running on iOS and Android). There are a number of design decisions that relate to the structure of the network infrastructure that supports the application based on the infrastructure required, the topology of the network, etc. The data collection strategies of the components is something that was focused on a lot in prior design spaces based on MDEs, but is not evident in current EUSC applications. Finally we have the context of the application, which is the context in which it is used, and the context in which the composites are executed.

The technologies that support composition applications are hard to identify in available applications, and as such are mainly gathered from prior design spaces in the domain and existing research literature. The technologies in the design space cover aspects such as the format of the data source, the communication protocol between the components, the language in which the application is implemented, and the language by which the components are described.

The last group of design decisions within the structural category relate to the types of component that the application supports. These types relate to how the component itself operates – in response to some action (triggers), or when it is told to execute (actions), as well as the support for and types of inputs to and outputs from that component.

4.4.4 Entity Category

The entity category is based on the entities that the application represents to the user when they are using it. There are aspects that would normally fit within either the functional or non-functional categories, based on interactions with entities and the attributes that these entities present, respectively. In our case, the entities that the application represents are all services, either components or composites. We felt that it was important to keep the information relating to the services in the application together, than to keep the traditional split that comes in requirements specifications.

The interactions that users can have with the services in the application are split into interactions with the composites that have been created using the application, and the components that were used to create them. Component interactions include aspects such as how the component can be activated and subsequently used in composition, and community aspects such as rating and tagging components to inform other users of the application. Composite interactions have similar community-based features including writing reviews, as well as how the composites can be managed.

The attributes that are presented by the components are meant to be mostly independent of

their representation (hence the separation from the non-functional category), but may not be completely independent. For both components and composites, they are split into functional attributes, popularity-based attributes, and other attributes. Composites may also present attributes of the components that are contained within them. Functional service attributes (for both components and composites) are the attributes that the user might use to work out what the service does, and how they might want to use it. The attributes presented by composites and components are very similar, although obviously the needs of the user are different in each case. Popularity attributes are also similar across components and composites, in that they are effectively a proxy for a quality measure that can be associated with the service. They are particularly important for composites as there is a much greater scope for similar services as they may be created by normal users, rather than the developer of the application. Other attributes are effectively the same across components and composites, are simply the attributes that we identified that would not be associated with popularity or functionality. For example, the copyright owner of the service, or its version number.

The next section focuses on an evaluation of the method that we used to create the explicit design space. Since using design spaces is heavily reliant on tool support, our discussion of how our design space could be used to support designers.

4.5 Design Space Creation Method Evaluation

One of the objectives of this chapter is to create and describe a generalisable method for creating an explicit design space for a given domain. We demonstrated this method for EUSC. In this section, we will analyse how each of the stages of the method affected the space, in terms of size and structure. We used the size and structure of the design space to motivate the discussion of how each stage of the method affected the design space as these are the only indicators of the changes to the design space that are available to us that are domain agnostic.

A summary of the changes in size of each of the design space categories at each stage are shown in Table 4.9. This data is summarised graphically in Figure 4-4.

Table 4.9: The total numbers of decisions and solutions added at each stage of the method.

	Functional		**Non-Functional**		**Structural**		**Entity**		**Totals**		**Change**	
	Decision	Solution	Decision	Solution	Decision	Solution	Decision	Solution	Decision	Solution	Decision	Solution
DS Collation	26	64	20	47	26	80	0	0	**72**	**191**	+72	+ 191
Literature Review	30	63	23	42	33	88	0	0	**86**	**193**	+14	+2
App Review 1 - Iteration 1	31	75	29	69	35	101	8	57	**103**	**302**	+17	+9
App Review 1 - Iteration 2	37	90	29	74	38	104	8	58	**112**	**326**	+9	+24
Requirements	42	108	32	79	41	112	15	84	**130**	**383**	+18	+37
Design Study	49	134	37	89	42	120	15	87	**143**	**430**	+13	+47
App Review 2 - Iteration 1	56	160	37	93	42	120	15	89	**150**	**462**	+7	+32

We will use the changes in size to motivate a discussion of each stage, and why it had the effect that it did. We will then be in a position to make recommendations as to which of the

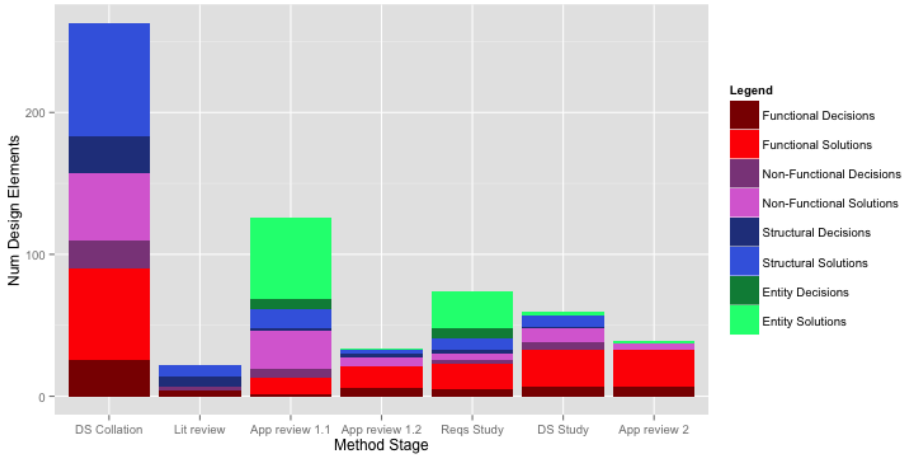


Figure 4-4: The size and structure of the EUSC design space at each stage of the method.

stages had more of an impact on the space, and hence how the method could be carried out more effectively in future and in other domains.

The design space collation stage had the largest influence on our design space, in that it provided both the base structure of the design space (although missing the entity category), as well as a large number of design elements across it. This stage of the method is relatively low in terms of time investment required to perform, although it has a much greater resource requirement. That is, the prior design spaces to be collated need to have been created by other researchers or designers working in the domain.

The general literature review of SC research added relatively few design elements to the design space, particularly when compared with the other literature-based stage – the design space collation stage. This is because of the large number of design elements and design topics that were identified in the literature review had already been added at the design space collation stage. If the design space had not been influenced by prior design spaces to such a great extent, then a review of the literature would have had much more impact on the design space as a whole. Thus, a literature review is likely to be much more effective for explicit design spaces being created in domains where examples of existing design spaces are not available explicitly.

The first application review in our method had a large impact on the design space, although not as strong an impact as the design space collation stage. It also became necessary to create another category, based on both functional and non-functional aspects of the services (more generally, entities). We created this separate category because of the very large number of properties that we identified that were related to the representation of, and interacting with the services in the composition process.

The iterative nature of our application review method means that we are able to repeat the

review in a robust manner where any additions made in an iteration i could be checked against all of the other applications in iteration $i + 1$. When carrying out this review it became clear that the changes made to the design space decreased with each iteration, so we elected to stop only when no changes were observed. The application review had a higher time cost than either the design space collation stage or the literature review, and can only be carried out in a domain in which similar applications already exist.

The requirements gathering stage of the method added roughly as many design elements to the design space as the initial iterations of the application review. Although a lot of the requirements that were gathered in this step were already present in the design space after the earlier stages, which is an obvious consequence of performing this stage of the method later. The requirements gathering stage has by far the highest time cost of all of the stages discussed so far due to the time cost of both the requirements engineer and the 10 participants who took part in the study.

Similarly, the design study added a number of decisions and solutions to the design space, but had a relatively low impact compared to the other stages – with the exception of the literature review. Given the size of the design space already, it is unsurprising that this stage did not have much of an impact.

The final stage of our method was to perform another application review using the same method as before, but with a larger set of applications, and a larger number of design elements in the design space than when the first application review was performed (i.e. those added or refined in the subsequent stages). The second application review did not identify as many design elements as the first, but this is to be expected given the number of design elements that we had already identified. We believe that this final application review was valuable because it incorporated the design choices made in new applications, as well as re-reviewing the applications from the original review and comparing them against the new design decisions and solutions suggested by the participants of our requirements gathering and design space studies.

Limitations

The most obvious limitation of the method is its intensity in terms of both time and resources – different stages of the method differ across both of these potential costs, but as a whole the method is very intensive. We discuss the limitations of each of the stages of the method separately, before considering the limitations of the whole method.

The main limitation of the design space collation stage is the requirement for other researchers, software engineers, or designers to have created design spaces in the domain of interest – or related/sub-domains. Obviously if there are no design spaces already in the domain, then there is nothing to be collated. In this case, the creator of the design space would need to use the general literature review and impose a base structure to the design

space themselves.

Our literature review was made less effective because a large proportion of the research on SC that was relevant to the design space had been covered in the design space collation stage. However, given that it is unlikely there would be a wealth of prior design spaces in other domains, the literature review would likely be more effective in other domains. Discovering and categorising topics from within the literature would have a higher level of required workload than the collation of design spaces.

The application review section of our methodology has one obvious disadvantage: design features that are not visible to the end-user of the application, and hence the reviewer of the application, cannot easily be assessed from only application use. Architectural decisions and solutions are not always evident from simply using an application, so the application review is less likely to identify such design elements, and much more likely to identify elements within our other categories. The only remedy to this would be to use secondary sources to identify aspects that are not visible to the end-user of each application.

The requirements gathering study and the design study both had the limitation of being very intensive in terms of time. The researcher or requirements engineer needs to design, carry out and analyse the results of the studies, and the participants of the studies are also required to give their time. Participant-facing studies are less intensive in terms of the resources that are required to undertake them, and any resources that are lacking can be created by the researcher prior to running the study.

The method as a whole also comes with an obvious limitation: diminishing returns. That is, the more that was added to the design space, the less effect the later stages had on the contents of the design space. This was evident in the application review, the requirements gathering study, and the design study because we could see the number of design elements that would have been added if they were not already present. We noted that the application review also benefits from an increase in the contents of the design space, which goes to some length to mitigate this issue.

The limitations highlighted here should be reflected upon for future applications of this method in other domains. However, there should not be a requirement to perform this method again in the EUSC domain, since we provide a comprehensive explicit design space for EUSC applications in Appendix D.

Generalisability

In this section, we discuss how generalisable the method is as it is presented here, and what could be done to make it more generalisable in future so that the method can be used to create explicit design spaces in other domains. Our discussion will focus on the generalisability of each stage of the method in isolation, before moving onto the generalisability of the method

as a whole.

The design space collation section is probably the least generalisable of any of the stages of the method due to its reliance on other researchers or designers having created design spaces in the chosen domain already. Thus there are two cases for generalisability of this stage in a given domain: either prior design spaces exist in that domain, in which case the design space collation stage is generalisable; or prior design spaces do not exist, in which case it is not.

The generalisability of the literature review is also contingent on the research that already exists in the domain. However, given the body of research that is available, it is likely that there will be some prior research in the domain, or a closely related one. Furthermore, the effectiveness of our literature review was reduced because of the effectiveness of the design space collation stage, which would not be the case in a domain where a design space collation stage could not be carried out.

In our implementation of the method, the application review had nearly as strong an effect on the design space as the design space collation stage. This could be carried out in any domain in which there are applications that can be reviewed, or similar domains if no applications are available in the chosen domain. For instance, if there were no EUSC applications that could be reviewed, we might instead have looked at business workflow applications, and app stores. Domains that are too mature may also suffer from a problem of having too many applications to review.

The requirements gathering stage of the method as we performed it was specialised to work in our chosen domain. That is, we used an established method for gathering requirements based on scenarios, and customised it to make it more applicable to an end-user focused domain. This was a very time intensive process, since we needed to design and carry out the study, as well as transcribing and analysing participants responses before using these responses to elicit requirements. These requirements could then be added to the design space in the relevant position. Clearly if the resources are not available in the domain to perform the requirements engineering method that we used, then another more appropriate method could be selected instead.

The generalisability of the whole method is based on the availability of the resources in the new domain to be considered. That is, the design space collation stage, literature review, and application review, all have strict requirements on the resources that are available before the stage can be carried out. The participant-facing stages, however, do not have such stringent requirements. We have identified that there is a trade off between the required time expenditure and the resources that are available: the time expenditure for stages requiring few resources is high, and low for stages with stringent resource requirements. The time requirement for stages does not affect the generalisability of the stage, but stages with a high expenditure might be unattractive to designers.

4.6 Discussion

Each of the stages of our method improves on methods described in other design space creation methods within the domain. This is demonstrated by the comprehensive description of each stage that we provide, as well as providing discussion as to the relative effect of each of the stages that make up the method. However, the strength of the method comes from the integration of these stages together to create a single design space. The main disadvantage compared to other methods is the large time cost, although stages could be removed as necessary to reduce this cost.

Our method contained three stages that were each used separately by other authors whilst they created their design spaces: the literature review, the application review, and requirements gathering. Our literature review was used mainly to fill in the gaps left by the design space collation stage, whereas [Mehandjiev and De Angeli, 2012] used it to provide the structure for their concrete design space by gathering together a set of concepts relating to both EUSC and EUD.

The design space collation stage and the design study have not been used before. The design space collation stage is effectively a combination of the approaches use by these authors to create their design spaces: application reviews, and literature reviews. As we discussed this stage is dependent on the resources that are available in the domain – in particular that other researchers have created design spaces in the domain. If this stage cannot be carried out, the literature review needs to impose the structure upon the design space, and should have more of an effect on the size and structure of the space as a whole.

Our application review is much more comprehensive than those in the previous design space creation methods. We felt in necessary to derive and describe a comprehensive and repeatable method within this stage. We based our method on heuristics identified by MacLean and McKerlie [MacLean et al., 1991], as well as maximising the identified design elements by incorporating iteration. We also performed two rounds of the application review at different stages of the method to maximise its effectiveness and to account for newly released EUSC applications.

The stages that make up the application review mean that the existing contents of the design space motivate new elements, in that solutions present in applications being reviewed are likely to be missed if the design space doesn't contain the corresponding design decision. Hence completing the application review a second time with the same set of applications could add design elements to the design space that were not identified the first time around. Furthermore, it means that any new applications released in between reviews can have their design choices added to the design space. Performing the application review twice highlights the potential to perform a second literature review to identify new literature that is created in the domain. However, the literature does not have the same property as the application review where the contents of the design space can motivate new design elements being identified

when the same applications are reviewed.

Requirements gathering was used by [Albinola et al., 2009] to create a design space, reinforcing the link between requirements and design spaces postulated by [Geyer, 2000]. However, [Albinola et al., 2009] do not disclose from where the requirements for their application were derived, so we felt it was important to describe a method that included sufficient detail to allow others to understand how requirements could be derived.

The design space tool study in Chapter 6 also identified a number of historically novel design ideas within the domain. The design space study required a design space to have already been created for the domain, and the time requirement of carrying the study out was very high.

The stages of the method were one of two different types: either reusing knowledge that already exists in the domain, or participant-facing stages. Knowledge re-use encompassed the design space collation stage, the literature review, and the application review. All three of these stages are relatively low in terms of cost, but require certain resources to already exist in the domain. The design space collation stage in particular is much less likely to be applicable across a number of domains.

The participant-facing stages were the interviews or studies: the requirements gathering stage and the design study. These have *much* higher requirements in terms of time cost, but lower requirements in terms of available resources in the domain. This stage also has the advantage that many participants can provide many different viewpoints on the design space, and were able to produce historically novel design entities.

The consequence of the different requirements for the different types of stage in our method means that different stages could be prioritised based on the resources that are available in the domain, or the time that is available to the designer or researcher. For instance, if the domain has a large amount of available resources then the designer or researcher can take advantage of these and reduce the time that is required to add to the design space, whereas if there are limited resources available in the domain then the researcher or designer's time costs will be much higher.

It is clear from the above evaluation that we can use the relationship between the decisions and solutions in order to identify new possibilities for solutions to these decisions. Again, participant facing aspects are helpful for this – either presenting them with a decision and possible solutions and asking them for more solutions, or by presenting them with a decision and no possible solutions and then asking them to generate new solutions. We will see how this can be applied in the empirical evaluation of the design tool for design generation in Chapter 6.

4.7 Chapter Summary

This chapter addresses part of our second research goal: **RG2**. **Create and evaluate a design space for EUSC applications.** In the chapter, we documented the creation of the design space, as well as providing an overview of its contents. The whole design space is very large, and is presented in Appendix D.

Before creating our design space, we provide some clarification regarding the terminology that is used to describe design spaces so that we are in a position to adequately describe what we have created, and be able to relate it to design spaces that have been created both in the EUSC domain, and in other domains. We provide two definitions of types of design space: implicit (conceptual, theoretically infinite) and explicit (concrete, finite and usable), as well as describing how these design spaces can be used, and where in the software engineering process we believe each of the uses is most suitable.

Our review of prior design spaces did not identify any well-specified methods that detailed how an explicit design space could be created for a given domain. Even the design spaces we found in the EUSC domain provided little insight into their creation, instead focusing on a discussion of their contents. This motivated us to devise our own method for creating an explicit design space, which we demonstrated in the creation of an explicit design space for EUSC applications. The method was made up the following distinct stages:

1. Initial collation of prior design spaces.
2. Literature review
3. Application review
4. Results of requirements analysis (Chapter 3)
5. Results of design tool evaluation (Chapter 6)

Whilst carrying out the method, we analysed the effect that each stage had, as well as discussing the resources (both time and practical resources) that were necessary to complete the stage. Following this analysis, we described the generalisability of the method to other domains, as well as being able to suggest what stages might be best to carry out given the resources that are available in the domain for which the explicit design space is being created.

After documenting its creation, we then provided an overview of the contents of our design space for EUSC applications by discussing the main topics that existed within each of the four categories of the design space: functional, non-functional, structural, and entity. We could not present the whole design space due to its size, and descriptions of the design decisions and corresponding solutions can be found in Appendix D.

This chapter addresses the design space creation aspect of our second research goal, but does not detail any evaluation. In order to be able to evaluate the explicit design space, we first need to address our third research goal and create a tool through the use of which we can evaluate the design space itself. The next chapter addresses the third research goal, and the

evaluation of the design space is presented in Chapter 6.

CHAPTER 5
THE DESIGN SPACE TOOL

5.1 Chapter Introduction

In Chapter 4, we documented the creation of an explicit design space for EUSC applications containing over 600 design elements. Clearly it would not be practical to manually interact with an artefact of this size. Indeed, [Baum et al., 2000] suggests that design spaces need software support in order to be used in practice by designers and software engineers, which prompted our final research goal to create and evaluate a tool that designers and software engineers can use to interact with design spaces.

We also discussed where design spaces can be used in the software engineering process: being used to profile existing domain applications, for the generation of new applications, and the validation stage for evaluating an application in the domain (this is reflected in Figure 4-1 in Chapter 4). We also drew parallels to design spaces and system modelling – a process that is said to occur between requirements gathering and system design [Sommerville, 2011].

In this chapter, we describe a design space tool that was created to help designers to interact with design spaces and use them in the design process. Our design space tool supports three main functions, which were identified from prior work on design spaces [Gooch, 2013, MacLean et al., 1991, Lane, 1996, Aghaee et al., 2012]:

1. Supporting the creation of explicit design spaces by allowing the user to add and connect together various design decisions and solutions that they have identified in the domain.
2. Recording the results of application profiling, where the user records the design choices made and decisions considered in applications in the domain, which may later be usable for evaluating the design of other applications in the domain.
3. Supporting the user in generating new designs for applications in the domain by recording their selection of design choices from the explicit design space as well as their own suggestions for design solutions.

We describe each of these three tasks in detail, and identify where they can be used in

software engineering. We also describe the module of the design space tool that corresponds to each of these tasks, and identify how the user of the design space tool interacts with each module. Since the processes of application profiling and design evaluation are very similar in terms of the stages that are required to carry them out, their tool support is amalgamated into a single module: the application profiling module.

To ensure that the design space tool supports these tasks, we perform an analytical evaluation to compare the task flow without tool support, and then compare this with the task flow using the tool. We are able to evaluate design space creation and application profiling effectively using these techniques. However, the task flow for design generation is not clear, and as such is not appropriate for this kind of evaluation. We demonstrate a prototype of the design space tool being used by 40 participants for design generation in Chapter 6.

Creating a tool to support the use of design spaces is an important step towards solving our research goals. We aim to identify how design spaces can be used to aid with the design process in software engineering, and this is impossible without the support of a design space tool. The tool as presented in this chapter operates using the explicit design space for EUSC applications that is documented in the previous chapter, and listed in Appendix D. However, the tool could be used to record and use an explicit design space for any domain.

The aim of this chapter is to address our third research goal: the creation and evaluation of a tool that can be used to provide support in the design process through the use of design spaces. This aim can be broken down into two of key objectives:

1. Create a design space tool that allows a designer to complete the following tasks:
   (a) Create an explicit design space in a domain.
   (b) Profile and evaluate the design choices made in applications in the chosen domain.
   (c) Generate new designs for software artefacts in the chosen domain.
2. Evaluate the support that the design space tool provides to designers performing these tasks.

5.2 Design Space Tool Overview

The Design Space Tool is a Web-based application that allows a designer to perform three key tasks related to explicit design spaces:

- **Design Space Creation:** Create an explicit design space by adding and connecting together a number of design decisions and corresponding solutions to these design decisions.
- **Application Profiling:** Profile existing applications in the domain of the design space (e.g. EUSC) to identify and evaluate the design choices made in the creation of these applications.

- **Design Generation:** Generate designs for new applications in the chosen domain (e.g. EUSC) by using the design space tool to select design choices from the existing explicit design space and contribute additional design choices that are not present in the explicit design space.

Each of these tasks is supported by its own module within the design space tool, and we provide a brief overview of the functionality provided in each module here, before going into further detail in Sections 5.4, 5.5 and 5.6.

The first module is the design space creation module, which allows users to create an explicit design space. When the user indicates that they wish to create a new design space, they are presented with four empty categories into which they can add design elements (functional, non-functional, structural, and entity). These design elements might be identified using a method such as the one described in Chapter 4.

The user might first identify a design decision in an existing application in the domain and add this decision to the design space (e.g. 'service discovery' in an EUSC application). The user could then connect this design decision to the relevant category of the design space (the functional design space in our example). They might then identify a potential solution to this decision (e.g. 'text-based search' as a solution to 'service discovery'), and add this solution to the design space. Finally, they would connect this solution to the existing design decision. The design space creation section of the design space tool is shown in Figure 5-1, and is described in more detail in Section 5.4.

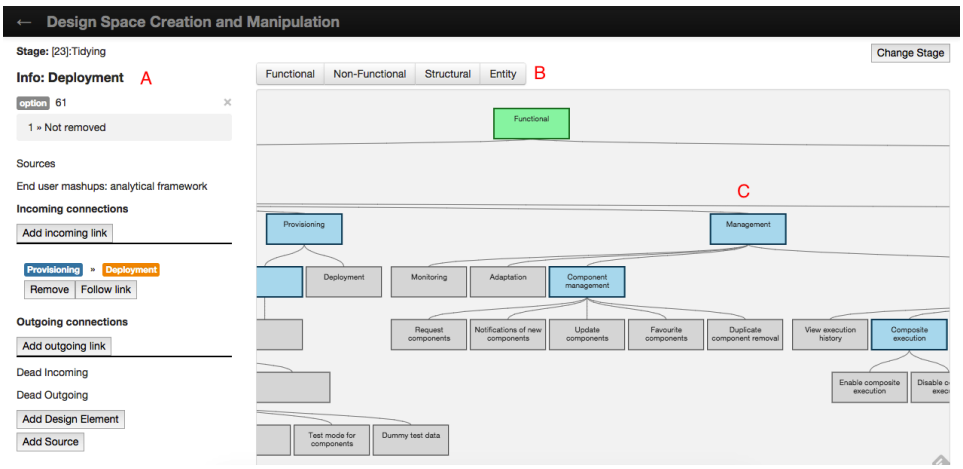


Figure 5-1: The design space creation module of the design space tool: A. The information pane; B. The category selector; C. The explicit design space.

The application profiling module of the design space tool allows the user to record the design choices that are made in applications in the domain (e.g. EUSC applications). The application profiling module requires that a design space has already been created using the design space creation module.

Users can add examples of applications to the profiling module, and then record the design choices that are made in that application. To indicate that a particular design choice has been made in a tool, the user selects the tool in the side bar of the design space tool, and then clicks the relevant design choices in the design space.

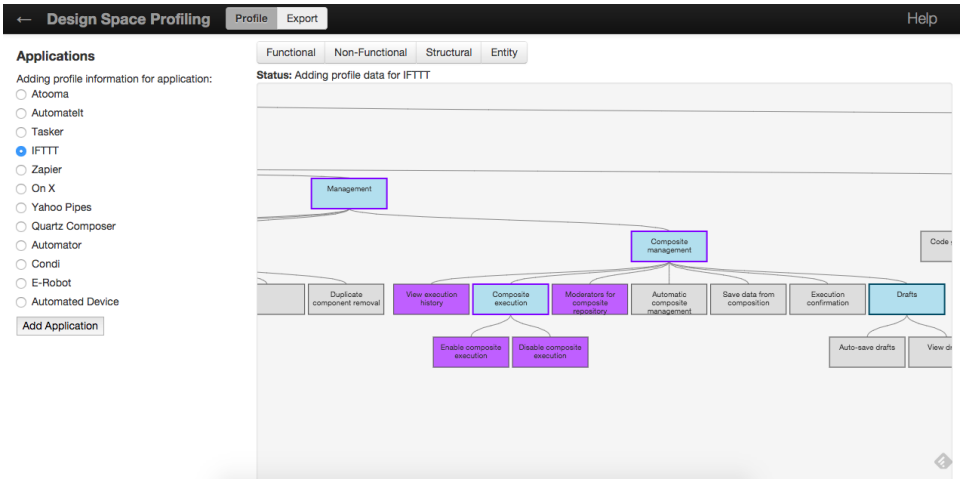


Figure 5-2: The application profiling module of the design space tool.

Figure 5-2 shows the profiling module of the design space tool whilst recording the design choices made in IFTTT. The profiling module also allows the designer to output the profile of an application or multiple applications. These features are discussed in more detail in Section 5.5.

The final module of the design space tool is the design generation module. This module allows a designer to generate and record a design with the aid of an explicit design space that has already been created using the design space tool. Designers can choose elements from the design space or suggest their own design choices. Each design choice that is made must also include a rationale indicating why that design choice has been made. The design can be exported upon completion. Since it is not clear exactly how the task of design generation is carried out by designers, we evaluate this module of the design tool empirically in Chapter 6. The design generation module is described in detail in Section 5.6.

5.3 Creating the Design Space Tool

Having given an overview of what the design space tool is and how it can be used, we will now describe how it was created. We first discuss the model upon which the design space is built, before describing the requirements for and design of the design space tool.

5.3.1 Design Space Model

The explicit design space is the main viewable component of each of the modules of the design tool, and our earlier discussions of the design space do not provide enough detail to be able to consider all aspects of the implementation of an explicit design space. This section discusses the model upon which the design space is built, in order to define the objects that need to be presented in the various representations of the design space inside the design space tool.

The model upon which our design space is built is based on the model presented by [Nowak and Pautasso, 2011] and subsequently used in [Aghaee et al., 2012]'s design space. There are three types of design element supported by the design space tool:

- **Category:** Broad categories into which the design decisions and solutions are grouped.
- **Decision:** Design decisions that can be made by an application in the given domain.
- **Solution:** Potential solutions to the design decisions in the model.

Each of these design elements must present its name (which should be short), and a description so that it can be understood by any other users (the description can be any length).

There are three relationships between the types of design element within the design space model. The relationship between decisions and solutions is clear: a decision is solved by one or more solutions. Decisions can also be grouped together underneath a 'parent' design decision. For instance, our model contains aspects of the EUSC life cycle (a parent decision), which is connected to a number of sub-decisions which represent stages within this life cycle. These sub-decisions can then be broken down into further sub-decisions as necessary. Categories can be broken down similarly into the high-level decisions that fit into that category.

There are also two properties that each of the design elements can have that are based upon aspects of our method for creating the design space: the source from which the design element was identified in initially (e.g. a particular piece of literature, or a particular application), and the stage in the method at which the design element was identified. The stage at which the design element was identified is the stage of our method that first found or suggested as the design space was created. The stage at which the design element was identified as no longer being necessary and removed from the design space was also recorded for design elements that did not make it through to the final stage of the design space creation method. For example, if decisions or solutions were split up or amalgamated.

The sources of the design elements are either applications in which the design element is first identified, or the piece of background literature in which the design element is suggested or identified. This property is only applicable to design elements that were identified either in the design space collation or literature review, or one of the iterations of the application review.

5.3.2 Requirements for the Design Space Tool

Across the creation of our explicit design space (Section 4.3), and our review of prior work on design spaces (Sections 2.9.3 and 4.2), we identified four key tasks that a design space tool would need to support, and hence four key functional requirements:

R1. The design space tool must allow users to create explicit design spaces. Users must be able to add design elements to design spaces in order to build an explicit design space in a given domain.

R2. The design space tool must allow users to record the results of application profiling. Users must be able to record which design choices are made in applications in the domain.

R3. The design space tool must allow users to evaluate existing applications in the domain. Users must be able to identify what choices have been made in a domain application and evaluate the success of this combination of design choices.

R4. The design space tool must support designers in generating designs. Users must be able to choose design choices from the explicit design space to use in their own, new design, as well as being able to suggest their own design choices that are not in the design space.

The creation of explicit design spaces was identified as being an important aspect of interacting with design spaces that needed support from our tool because of the effort required to manually create the structure of the design space and have it in a usable format after it has been created. We identified a number of features that would be required to support the creation of the design space, which will be described in detail in Section 5.4.

Application profiling is an activity that was identified by [Baum et al., 2000] (referred to as design space profiling by Baum et al., but renamed here for clarity), and has been used widely in our chosen domain of EUSC [Aghaee et al., 2012, Grammel and Storey, 2010, Minhas et al., 2012, Na et al., 2010, Mehandjiev and De Angeli, 2012, Albinola et al., 2009, Fischer et al., 2009, Pietschmann et al., 2010, Brønsted et al., 2010, Tuchinda et al., 2011]. Application profiling is the process of reviewing applications to identify the design decisions that are considered, and the solutions that are chosen across these applications. Tool support can be provided in application profiling by helping with the process of recording the design choices that are made a given application. The process of reviewing these applications must still be a manual one, however the representation of the explicit design space means that the designer performing profiling can process it systematically, and hence reduce the likelihood of errors in recording the chosen design solutions.

Design evaluation relies on using value judgements associated with particular design choices, or collections of design choices, to be able to say whether a particular design is ‘good’, or ‘bad’ (or, more likely, somewhere in between). Since the process of assigning this value to the design choices is unclear, as well as being specific to a single application, we only

consider the part of the task where the designer records the choices that are made in an application in the domain: which is simply the process of application profiling.

Exploring the effect of using design spaces in design within the software engineering process is one of the main aims of this thesis, and as such we needed to create a tool that can support the designer in using the design space in this way. Since at this stage we do not know exactly how a explicit design space can be used in the generation of new designs, it is difficult at this stage to know exactly what features might be required for creating designs. The support provided by the tool for using the design space in the design process will be discussed in more detail in Section 5.6.

Designing the tool based on these four requirements yielded three sections of the tool: design space creation, application profiling (support for design evaluation is also included in application profiling), and design generation.

Target User

Our design space tool is targeted at designers who have been given a task of designing a piece of software in a given domain, or are researching a domain for a potential design in that domain. Each of the sections of the tool could be used by a different designer in a team, or could be completed by a single designer. The activities that the tool supports must be carried out separately, so providing support for multiple activities at once is not necessary.

5.3.3 Design and Implementation

We chose to design the tool as a Web application due to the availability of graph libraries to render the explicit design space in JavaScript. The tool follows a Model View Controller (MVC) architecture, where the Model is a database back-end (the design space), and the View is based on the aforementioned JavaScript graph libraries and other associated HTML/CSS/JavaScript front-end representation. A controller then allows for interaction between the model and the view, as well as controlling the logic of the application.

Other factors that led to the choice of a Web application include a relatively short development time and the availability of open source data storage technologies. All three of the main functions of the tool have a similar design: a canvas that contains a representation of a category of the design space, with a side bar containing information or settings for the design space.

The tool was designed to allow the user to create multiple design spaces across different domains so that a designer or design team can work on multiple domains at one time. The Design Space Tool is currently optimised for a single user working on a design space, although it is possible for a number of users to use the tool across different design spaces. The design of the tool does not prohibit multiple concurrent users from working on the same

space, but since supporting multiple concurrent users was not an explicit aim of the design tool, this may lead to inconsistencies in the design space or the design being generated.

The back-end of the tool is implemented in PHP, which is connected to a MySQL database that stores all of the information regarding the design space model, application profiling, and the design decisions made in the design phase. PHP is used to transform the data in the database to JSON, where it can be easily interpreted by the JavaScript-driven front-end.The front-end is built using HTML/CSS/JavaScript, using three libraries: jQuery[18], Bootstrap[19] and JavaScript InfoVis Toolkit (JIT)[20]. jQuery and Bootstrap allow us to create a responsive Web application, and JIT allows us to represent a large explicit design space on a canvas that the user can navigate around, regardless of the size of the design space.

5.4 Design Space Creation in the Design Space Tool

The home page of the tool allows the user to create a new explicit design space, before they are given the choice of three different options as to how to interact with the newly created design space based on the three different modules identified earlier. The first feature is to add elements to the design space and subsequently manipulate the contents of the space into a given structure, which is discussed in this section. The second alternative is to record the design decisions and solutions that are chosen in various available tools in the domain, which is discussed in Section 5.5. Finally, the user can choose to create a new design by selecting alternatives in the design space presented by the design space tool, or choose to add their own decisions or solutions to the design. This will be discussed in Section 5.6.

The first module of the design space tool allows the user to manipulate the elements within the design space. When created, each design space is just a set of four main categories: functional, non-functional, structural, and entity. The user of the design space tool can then add design elements into a hierarchy within these categories. Software engineers can use the design space creation module to help them to create an explicit design space in a given domain. No restrictions are imposed on the domain that can be chosen, and the designer can add any collection of decisions or solutions

In the design space creation stage, the designer adds and connects together design elements. We provide designers with a mechanism for recording and connecting together the design decisions and corresponding solutions that can be used in conjunction with a design space creation method – such as the one described in the previous chapter – to create a design space. The explicit design space created in this module can be exported and used in the other modules of the design space tool: for application profiling and design generation.

[18] http://jquery.com
[19] http://getbootstrap.com
[20] http://thejit.org

5.4.1 Design Space Creation Features

New design elements need to be added to the design space creation tool before they can be linked to a parent design element. Each new element in the explicit design space needs to have the following properties:

- **Name**: A short name for the design element. This should be fewer than five words in order to be presented in the tree representation of the explicit design space.
- **Description**: A longer description of the design element. This can be of arbitrary length.
- **Type**: Whether the design element is a decision or a solution.
- **Source**: The application or literature in which the design element was first identified. Sources are selected from a list which can be populated elsewhere in the tool.
- **Stage added**: The stage at which the design element was identified in the method. This is recorded automatically based on what stage the user reports they are currently undertaking.
- **Stage removed**: The stage at which the design element was identified as being no longer necessary. As with stage added, this is recorded automatically by the tool.

Once a design element has been added to the design space, it can then be linked with other design elements in the explicit design space. After design elements have been added, a number of properties of the design element can subsequently be changed (as well as modifying links to other design elements).

The source of the design element, and the stage at which it was added (or removed) are important for traceability. This means that when designs are being generated, the designer is able to identify where particular design elements originated quickly.

Consider the example design solution where users of the EUSC application are able to request new components be added to it:

- **Name**: Request components
- **Description**: The user of the EUSC application is able to request new components to be added to the EUSC application.
- **Type**: Solution
- **Source**: User study
- **Stage added**: Requirements gathering study
- **Stage removed**: N/A

Stages and Sources The design space tool also keeps track of what stage of the method the user is currently performing (this is applicable to our design space creation method described in Section 4.3). Stages can be added to the tool when the user starts that stage, and if the user navigates away, the design space tool will reload the latest stage that has been added.

For example, the 'request components' design solution was added in the requirements gathering stage of our design space creation method. To identify this, the user would add a new stage named 'requirements gathering study' and then select this stage. Any design elements they added after this point would be assigned to the requirements gathering study stage of the method. The stage at which a design element is removed is also recorded automatically.

The first stage is created by default when the design space is initially created. Thus, if the user's method for creating their explicit design space is not split into stages then the tool can simply be left to add design elements at this default stage. This also means that users of the design space tool are not restricted to using our design space creation method.

The sources of the design elements are also recorded, and need to be added manually before they can be associated with design elements. Sources are only applicable to design elements that are identified from literature or in the application review, participant-facing stages are not considered. Sources are added in a similar way to design elements, but have different properties that can be associated with them:

- **Name** – The name of the source, either the name of the paper or book if it was from the literature, or the name of the application if the source is an application.
- **Author or Creator** – The author of the literature, or creator of the application.
- **Type** – The 'type' of source: either background literature or a application from the domain.

Recording the stage at which a design element was added or removed allows designers to identify how the design space changed over the course of the method, perhaps to identify stages that they may want to perform again. Sources are recorded for traceability if a design element is chosen to be used in a design that is generated using the design space tool.

Links Between Design Elements Once design elements have been added to the tool, they can be positioned within the design space, and either be added as an incoming link to a design element, or an outgoing link. Links can also be removed to remove or re-position design elements within the design space.

5.4.2 Design Space Creation Output

Once the user has added elements and links to the design space, they can then navigate around the design space by clicking and dragging on the canvas. The user can then interact with the design elements in the tree in two ways:

1. **Left-click:** Left-clicking on a design element brings up information about that element. The properties of the element (described above), and its links with other elements are shown in the sidebar.

2. **Right-click:** Right-clicking on a design element expands or contracts the tree below that design element.

5.5 Application Profiling in the Design Space Tool

Section 2.9.3 showed that by far the most popular method for evaluating explicit design spaces was to profile existing applications in the domain and identify the design decisions and solutions that were considered within them (this process is described in detail in Section 2.9.3). Whilst we do not consider this a method for evaluating a design space *per se*, we believe it is an important component of design space analysis because it allows a designer to see what design choices have been made in a given application at a glance.

As we discussed in the previous section, design evaluation is a task that is related to application profiling that we aim to support with our design space tool. The only aspect in which application profiling and design evaluation differ is the value judgement that is made about particular design choices or collections of design choices in the design of a given application. Since it is not clear how this value is ascribed, and that the value judgements will differ per application or design that is being evaluated, we chose not to reflect this side of the process. Thus, the only part of this process that we need to support is recording the design choices made in the application being evaluated, which is identical to the process of application profiling.

To support the process of application profiling, we provided a simple mechanism for recording which design decisions have been considered in the design of an EUSC application, as well as the particular solutions that were chosen in that application. It is then possible to view the decisions and solutions that have been implemented in a particular application, or see the relative popularity of decisions and solutions across all of the profiled applications. Application profiling is another important aspect of tool support in design spaces as it is a very involved process and would require a high time investment if completed manually.

The application profiling stage records the design choices that are made in applications in the domain, and this information can be used in later stages of the design process, either in the evaluation of existing designs, or within the design generation module of the design space tool.

5.5.1 Application Profiling: Uses

We identified that there are four main uses for the results of an application profiling activity, three that have been used in prior work in EUSC design spaces, and one that has not:

1. Design evaluation, both of individual designs and trends in design elements that are frequently chosen together [Aghaee et al., 2012, Grammel and Storey, 2010, Minhas

et al., 2012, Brønsted et al., 2010, Pietschmann et al., 2010].
2. Evaluation of the explicit design space [Aghaee et al., 2012, Grammel and Storey, 2010, Minhas et al., 2012, Brønsted et al., 2010, Pietschmann et al., 2010].
3. Identifying areas of research within the domain [Mehandjiev and De Angeli, 2012].
4. Value ascription to design elements and combinations thereof.

The most common use of application profiling in prior work in design spaces in EUSC is to evaluate designs of EUSC applications [Aghaee et al., 2012, Grammel and Storey, 2010, Minhas et al., 2012, Brønsted et al., 2010, Pietschmann et al., 2010]. The implication provided by these works is that by knowing all of the design choices that have been made in a given application means that it is easier to come to some conclusion about the quality of the design of that EUSC application, although it is not clear how.

Conversely, design space profiling can be used to demonstrate whether a explicit design space adequately describes a domain. This is analogous to performing the application review stage of our method – any design elements that are not present in the explicit design space can be added.

[Mehandjiev and De Angeli, 2012] invited researchers to profile their favourite application in order to identify aspects of applications that are less well researched, and hence identify aspects of EUD in EUSC applications that require further research. A similar approach could be used to identify future work in any given domain using the design space tool for application profiling.

[Gooch, 2013] identified generation and evaluation as two potential uses for design spaces, but in order to be able to carry out either of these tasks, there needs to be some kind of value associated with elements in the design space, or value associated with combinations of design elements. The value could be assigned implicitly by the designer, or explicitly and recorded in the tool. The application profiling activity could be used to assign the value to design elements by identifying which decisions are considered most frequently, or which solutions were chosen in combination with one another and recorded explicitly using the design space tool. Once the design elements have been ascribed with some value, they can then be used to evaluate other applications in the domain, or generate new designs. We performed application profiling on each of the EUSC applications that were used in the application review of our design space creation method.

5.5.2 Application Profiling Features

Application profiling relies on identifying the design choices made in an application in a given domain, and the design tool can be used to record which design choices are made in the application. The tool provides two main features within profiling: recording the design choices made in the application that was profiled, and outputting the results of the profiling exercise.

Application Profiling Input

Applications to be profiled are shown in a radio group list in the sidebar. The application currently being profiled can be selected by clicking an element of this list. The design space is then updated to reflect the current profiling data for that application. By default, these applications are the same sources as those that are added in the design space creation section of the tool. Alternatively, new applications can be added as sources through a dialog in this section of the tool. Sources added here are automatically identified as applications rather than literature, and can be selected immediately once they have been added.

The main feature of this section of the tool is to select elements in the design space to indicate that they have been considered in the chosen application. Once an application has been selected in the sidebar, clicking any of the elements will indicate that it has been chosen in the application in the domain, and the design element will be highlighted accordingly. When a design element is selected, all parent design elements are also selected since by the hierarchical design of the design space, its parent decisions will also have been considered.

Once a design element has been selected (either manually or automatically), it can manually be removed by clicking on it again; once removed, it will then be de-highlighted.

Application Profiling Output

Once the design space tool has been used to enter the design profile for a given EUSC application, it can output the results of the profile for this EUSC application, or a collation of the profiles of all of the EUSC applications that have been profiled using the design space tool in a single explicits design space. It can also provide the profile or collated profile for a sub-section of the design space, which can be selected by restricting it to a single decision (at any level within the design space hierarchy). The result of the profile output can either be in the form of a table (.csv), or a tree (.gv), the relative merits of which are discussed in the next section.

To output the results of a profile, the tool can be switched to output mode. In this mode, more than one application can be selected at once within the action bar, and the profiling results for the selected tools are amalgamated and displayed in the design space hierarchy representation of the tool. Figure 5-3 shows the output view of the application profiling exercise for Atooma, AutomateIt and Tasker. The user can then optionally choose the root node for the output file, as well as the format of the output file from the action sidebar.

The tool allows the user to view the results of application profiling within the tool as well as being able to export the results so that these results can be used within the design process whether the designer is using the design space tool, or they are using the data elsewhere.

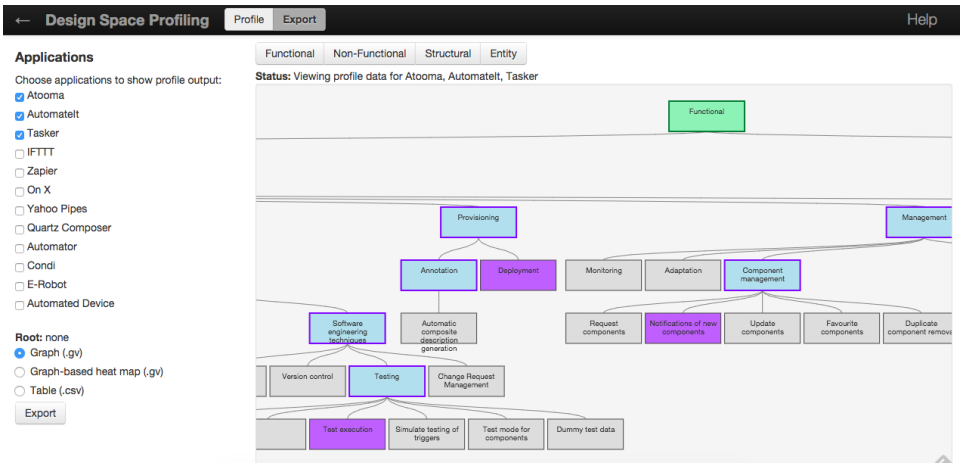


Figure 5-3: The output view of the application profiling module.

5.5.3 Application Profiling Results

In this section, we present some of the results of profiling some of the EUSC applications used in our application review, using two different presentations methods. Prior work [Aghaee et al., 2012, Minhas et al., 2012, Pinelle et al., 2003, Brønsted et al., 2010] normally present the results of design space profiling activities in a tabular form (a feature matrix). However, we found in the design of our design space tool and profiling function, that a hierarchical representation was more useful since it presented the links between design elements as well as the elements themselves.

Application Profiling Output: Individual Application

One of the forms of output of the profiling activity that the tool can provide is the design choices that have been made in a single EUSC application. Figure 5-4 shows a subset of the design choices that relate to service discovery that were made in Atooma (an Android-based EUSC application that is described in Section 2.7). Design decisions are presented as rectangles, and potential solutions as rounded rectangles. Decisions that have been considered (i.e. the tool has a solution for these) and solutions that are chosen are highlighted in purple. It is also possible to present this output in a tabular form, but for the output of a single tool this does not make sense as it would only contain a single row.

Application Profiling Output: Multiple Applications

The design space tool can also produce profiling results for several applications at once. There are two options for representing this graphically: presenting all of the design elements present across all of the applications (e.g. a binary relation), or presenting the proportion of

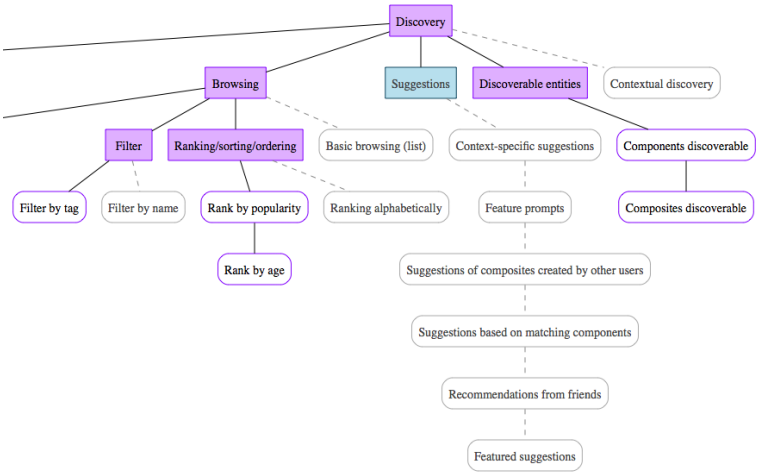


Figure 5-4: A sample output of the application profiling module of the Design Space Tool (applied to Atooma).

the selected applications that made this decision in the form of a heat map. Figures 5-5 and 5-6 show the decisions and solutions considered in AutomateIt and Atooma in both of the two representations, where Figure 5-5 shows all of the decisions made across both of the applications, and Figure 5-6 shows the same information in a heat map form.

In the binary representation, all design decisions that are considered are displayed in the same shade of purple, and the chosen solutions have a white background and purple border. Decisions that are not considered are grey, and solutions not chosen are white with a grey border. The heat map shows all decisions or solutions that have been considered in a given hue, and the saturation of the colour increases or decreases depending on the number of times that design element has been considered across the set of applications that are being profiled. In the case of Figure 5-6, design elements are shown in dark purple if they were considered in both Atooma and Automate it, light purple if they are only considered in one of the applications, or white if they are not considered in either application.

The heap map representation does not show either the number of applications or the specific application that have implemented each design choice because this representation is meant as an overview. If the designer wishes to view more information based on a heat map, they can output the same information in the form of a table.

Table 5.1 shows the same information as Figure 5-6, but in a tabular form instead of using a heat map. The tabular representation is the representation that has been used by all of the prior application profiling exercises within EUSC [Aghaee et al., 2012, Grammel and Storey, 2010, Na et al., 2010, Pietschmann et al., 2009, Brønsted et al., 2010].

We believe that the heat map-based representation can be advantageous because it gives a clearer overview of which design elements have been considered in applications previously

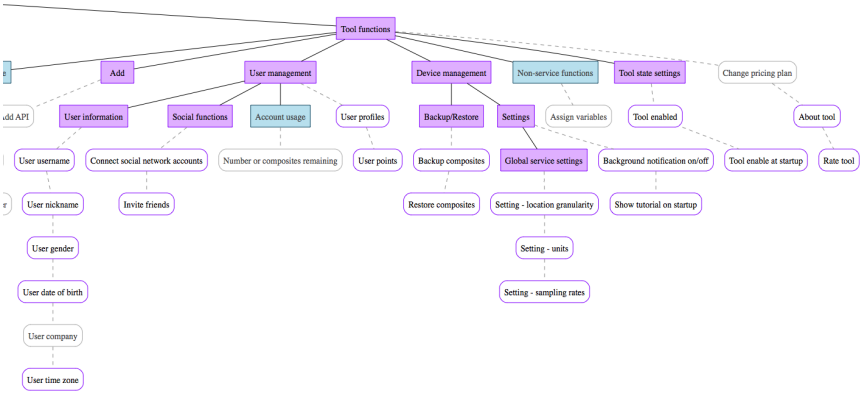


Figure 5-5: Binary profiling output for Atooma and AutomateIt.

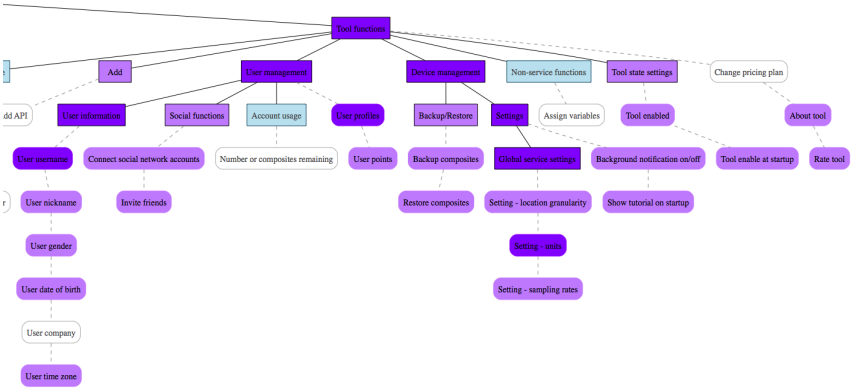


Figure 5-6: Heat map profiling output for Atooma and AutomateIt (Dark purple = chosen in both, light purple = chosen in one of the tools, white = not chosen).

and which have not. That is, if a large number of tools are considered, then design elements that are only considered in a single application would be very light in colour whereas all chosen design elements are shown in the same colour in the binary representation. Furthermore, the tabular representation does not adequately convey the relationships between the design elements, whereas the tree-based representation does. However, when trying to discern which design elements are more or less popular, the tabular representation can be more useful since values are explicitly quantified. These trade-offs meant that we implemented both representations as potential outputs for our profiler.

Table 5.1: The design elements considered in Atooma and AutomateIt relating to general tool functions (1 = chosen solution).

Application	Provide component wrapper	Add plug-ins	User profiles	User username	User nickname	User gender	Date of Birth	User company	User time zone	User points	Connect social network accounts	Invite friends	Number or composites remaining	Backup composites	Restore composites
Atooma		1	1							1	1				1
AutomateIt	1	1	1	1	1	1		1	1				1	1	

Reflecting on these two different presentation methods, we suggest that the heat map representation gives a much clearer view of which are the more and less popular design choices at a glance, but the tabular representation shows which tools made which design choice. The same can be said when the results of profiling more applications are considered: see Figure 5-7 and Table 5.2. When more profiled applications are considered, it becomes much more difficult to determine relative popularity between design choices in the heat map representation, which is not the case for the tabular representation. However, the tabular representation increases in size when more applications are considered whereas the tree-based representation does not.

This profiling information shows the design choices and combinations of design choices that have been chosen before in the domain. This information is useful because it shows designers the choices that they may want to consider since they have worked in other designs in the domain. Alternatively it can show choices that have not been used in existing designs, and hence those which they may want to consider as being innovative in the domain.

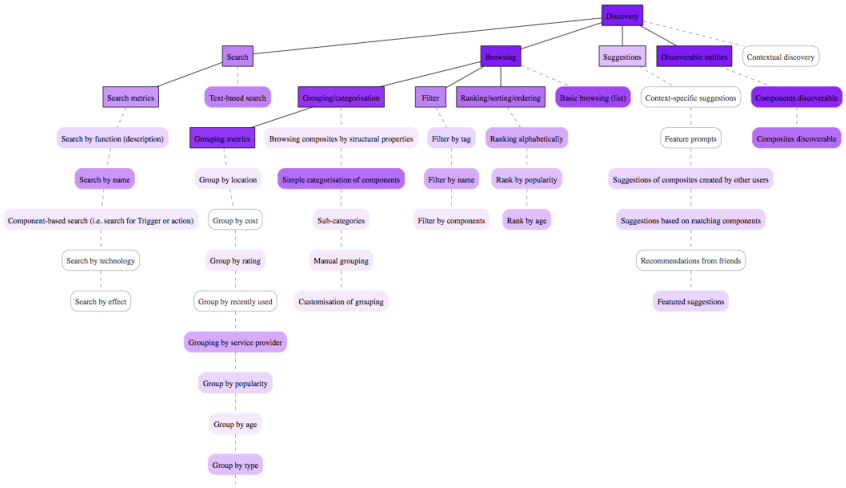


Figure 5-7: A 'heat map' representation of the application profiling results across the Tool function section of the Functional design space – applied to Atooma and AutomateIt.

Table 5.2: The results of application profiling across the discovery section of the functional design space

Application	Text-based search	Search by function (description)	Search by name	Component-based search (i.e. search for Trigger or action)	Search by technology	Search by effect	Browsing composites by structural properties	Simple categorisation of components	Sub-categories	Manual grouping	Customisation of grouping	Group by location	Group by cost	Group by rating	Group by recently used	Grouping by service provider	Group by popularity	Group by age	Group by type	Group by function	Group by network requirement	Group by featured/not	Filter by tag	Filter by name	Filter by components	Basic browsing (list)	Ranking alphabetically	Rank by popularity	Rank by age
Atooma								1	1							1	1	1		1			1					1	1
AutomateIt	1		1	1				1						1			1		1			1				1			
Tasker								1												1				1					
IFTTT	1	1	1					1								1						1	1		1			1	1
Zapier	1	1	1													1			1					1		1	1		
On{X}	1		1																	1						1		1	1
Yahoo Pipes	1																			1						1			
Quartz Composer																								1		1	1		
Automator	1		1					1								1				1				1		1	1		
Condi								1												1						1			
E-Robot							1	1		1	1	1							1	1						1	1		
Automated Device																										1			

5.6 Design Generation in the Design Space Tool

The final module of our design space tool is to allow designers to decide what elements of the design space (and other elements of their own choosing) should be included in a future design, and to allow them to record the rationale for the design decisions that are made in this process. This feature of the design space tool is meant to operate at the early stages of the design process, where the designer chooses or investigates functions and features that they might consider important in the chosen domain.

We feel that this could be particularly useful for designers who are unfamiliar with the domain in which the design is being generated, particularly since a single designer could do the research into the domain and create the design space, and then other members of the design team could decide on design elements from the design space.

Designers can use the design generation module to create a model of an application in the domain either by selecting design choices from the explicit design space, or by suggesting custom design solutions on their own. The results of this process can then be exported so that the designer has a record of the choices made in their design, along with associated rationale. It is not clear how this would be achieved with a design space but without the support of a tool, hence we are unable to evaluate how the tool supports the task flow already associated with this process.

5.6.1 Design Generation Features

The main feature of the design generation section of the design space tool is to allow the designer to choose design solutions for the application that they are trying to develop. Our tool supports this task in two ways: the designer can choose a potential design solution from the explicit design space, or they can add their own design solution. Both of these approaches to making a design choice allow the designer to record a rationale for the choice that they have made. We believe that this is important for traceability in the later stages in the design, since there is a record of why a particular solution was chosen over others – either as a prompt for the designer who made the choice, or for other members of the design or implementation team [Fischer and Shipman, 2013].

The user can navigate around the design space by dragging the canvas, which follows their mouse. They are also able to show or hide sub-elements of a particular design decision by left-clicking that decision. Note that the layout of the nodes in the tree does not change when this expansion or contraction occurs. The user is able to view more information about design elements in the design space hierarchy by right-clicking on them. Right clicking design elements brings up the dialog shown in Figure 5-8.

To choose a potential design solution that is part of the explicit design space, the designer must click on the potential solution in the explicit design space hierarchy. To find the design

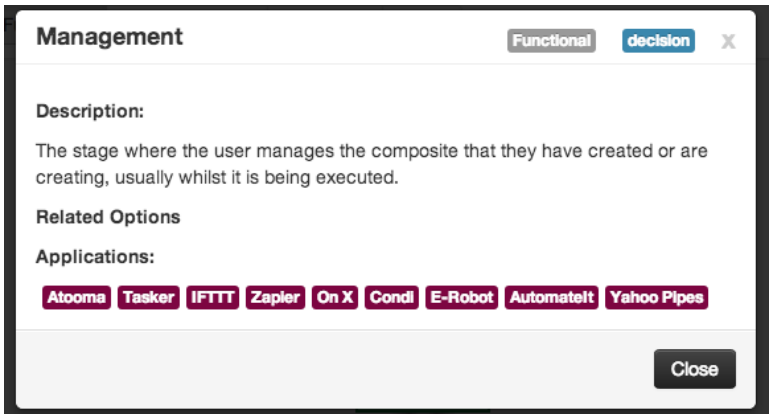


Figure 5-8: The information dialog for 'Management' in the explicit EUSC design space.

element that they want to choose, the user of the design space tool can change the category that they are viewing using the buttons above the design space canvas, as well as showing or hiding children of a design decision by clicking that decision. Potential solutions can be chosen by clicking them. Once they have clicked the potential solution, a dialog is shown where they must enter a rationale for choosing this solution. This dialog already contains the name of the potential solution, and a description of the solution so that they can be recorded as part of a design document.

To add their own custom design solution, the designer enters a name for their custom solution in the text box labelled "Add custom solution", and clicking "Add". Following this, the designer is presented with a similar dialog to when choosing a solution from the explicit design space. For custom solutions, the user must enter a description of this solution as well as a rationale.

Once the design has been completed, the user is able to export the design that they have created to get a record of what decisions have been made, and the reasons for all of these decisions. The design that is stored in the design space tool can also be reset.

The amount of the explicit design space hierarchy that is presented to the user can also be changed, hiding some potential solutions and design decisions in the hierarchy, based on the popularity of those design elements across the results of the design space profiling exercise. The information being shown in the design space hierarchy can be changed using two sliders at the top of the action panel: a slider controlling the proportion of decisions that are shown, and a slider controlling the proportion of potential solutions that are shown.

The sliders allow the designer to choose the percentage of each type of design element that is present based on the results of the design space profiling exercise. For instance, if the user selects 90% on the design decisions slider, all decisions are shown except for those that are considered in less than 10% of profiled tools. The same is applicable for design solutions.

There is an interaction between the design decision slider and the potential design solution

slider. That is, if a low percentage is chosen for the decision slider, then this will restrict the potential solutions that can be displayed. In this case, the potential solution slider presents the percentage of potential solutions, but restricted to those decisions that are being shown, dictated by the design decision slider. A screenshot of the design generation function of the design space tool is shown in Figure 5-9.

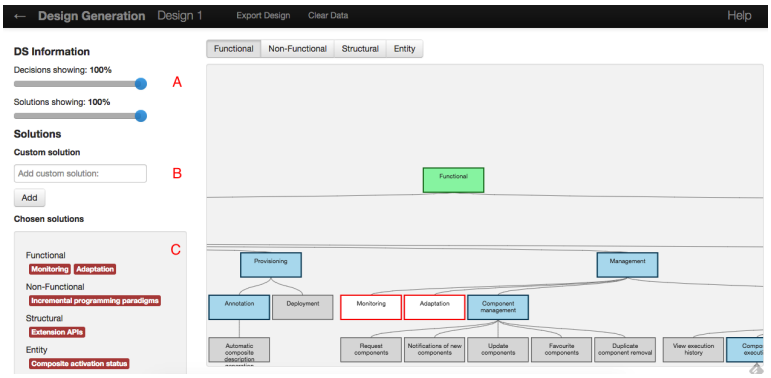


Figure 5-9: The design generation module of the design space tool: A. Element view slider, B. Custom design solution entry, C. Record of design so far.

5.6.2 Design Generation Output

The design generation section of the design space tool allows the user to output the design that they have created. The output presents the design as a list of the design choices that have been recorded in the tool, along with the descriptions and rationales for each solution that is selected. Design solutions are grouped together based on the category of the design space in which they are present. This information is also supplemented with profiling information, namely a list of tools in which the decision has been made. If the designer has generated any custom design solutions, they are presented below, including any links to design decisions that already exist in the explicit design space.

5.7 Analytical Evaluation of the Design Space Tool

One of the aims of this chapter is to evaluate whether the design space tool we created provides adequate support for the the three tasks we identified: design space creation, application profiling, and design generation. We decided to use analytical evaluation to view the task support before the design space tool is presented to users in any kind of empirical evaluation. Analytical methods seek to model current user behaviour, and to predict their future behaviour, and these methods can include heuristic evaluation, various different types of walkthrough, user, system and task modelling, and analytics [Rogers et al., 2011].

We identified the task-focused analytical methods as being most important to us as this is the aspect of the design space tool upon which our evaluation is focused. Thus, we chose to perform walkthroughs and task modelling. To perform these evaluations, we first had to identify how the task could be completed without the support of a tool, and then evaluate whether the tool provides the designer with an effective method for completing this task. The specific approaches we used were Hierarchical Task Analysis (HTA) and Cognitive Walkthrough (CW).

5.7.1 Task Analyses

Hierarchical Task Analysis (HTA) breaks a task down into sub-tasks (and breaking those sub-tasks down further), before providing an ordered plan to indicate how the tasks would be organised if the task were to actually be carried out [Rogers et al., 2011]. The feature of HTA that is of most important to us is that it is very useful in comparing alternative designs.

The tasks that HTA is used to evaluate must be simple, as the notation can become very complex and difficult to interpret if the task itself is complex [Rogers et al., 2011]. The tasks that we are evaluating are relatively simple since we focus on the tool support side of the task being undertaken and not the chosen method for performing the task external to the design space tool.

To determine whether our tool provides adequate support for the tasks, we performed HTA on the tasks of design space creation and application profiling before the tool was created, and compared these with HTAs carried out on the tool itself to evaluate the level of support that the tool provided.

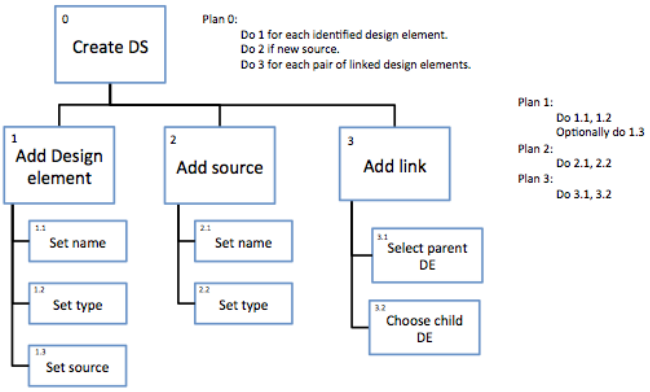


Figure 5-10: HTA for creating a explicit design space.

Figure 5-10 shows the HTA for the tasks that support the creation of a explicit design space, assuming some other method for identifying the design elements and the relationships between them (e.g. our method suggested in Section 4.3). Figure 5-11 shows the HTA for the tasks supporting the creation of a explicit design space in our design space tool.

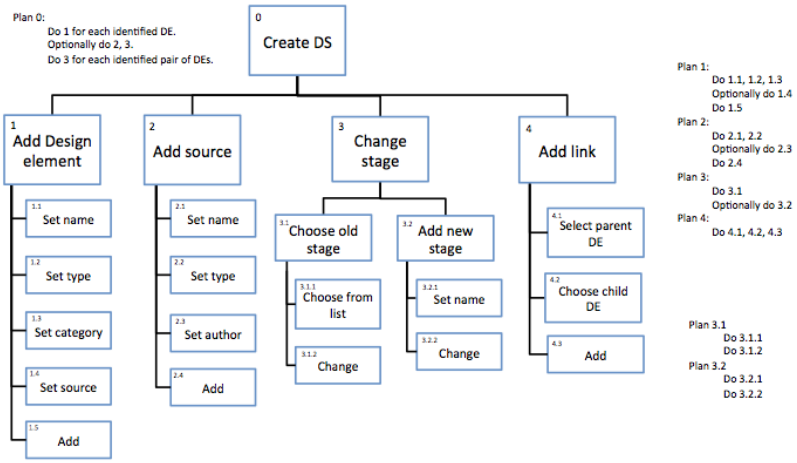


Figure 5-11: HTA for creating a explicit design space using the design space tool.

For the explicit design space creation task, the only difference between the task breakdowns was that the tool HTA included information regarding the stage of the method that the user is currently performing. This particular design element may be specific to our method since it is split into a number of stages, whereas other methods may not. This was not identified in the non-tool HTA because the value of the information regarding the stage at which each design element was added is not clear.

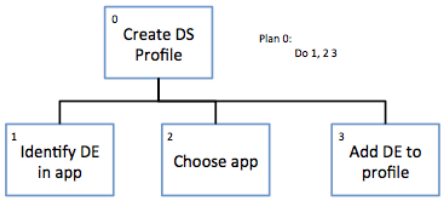


Figure 5-12: HTA for completing an application profile.

Figure 5-12 shows the HTA for the tasks that support recording the results of application profiling, assuming the user has some method for identifying design elements that are chosen in a particular application. Figure 5-13 shows that HTA for the tasks supporting recording the results of application profiling that are provided in our design space tool.

For recording the results of application profiling, our tool broke down the task of adding design elements to the profile into finer-grained tasks, as well as identifying a further task that was important in application profiling: exporting the application profile so that it can be used. Whilst this is not necessarily part of the application profiling process, it is a useful aspect of tool support for design spaces. Designers may use these exported profiles in other aspects of the design process, or outside of the design space tool.

We were unable to perform this comparison for design creation because there is no defined

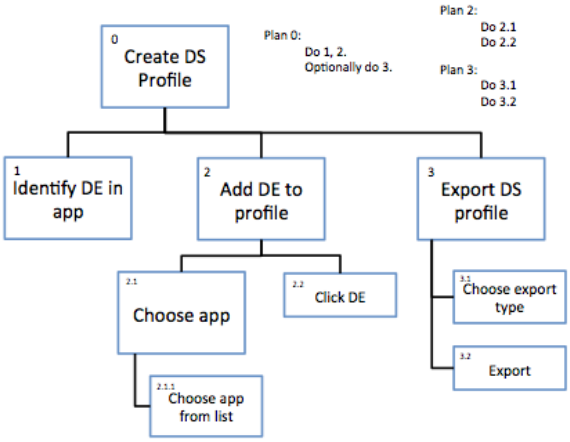


Figure 5-13: HTA for completing an application profile using the design space tool.

process as to how designs can be created using a explicit design space without the aid of the design space tool. The HTA for design creation using the design space tool is shown in Figure 5-14.

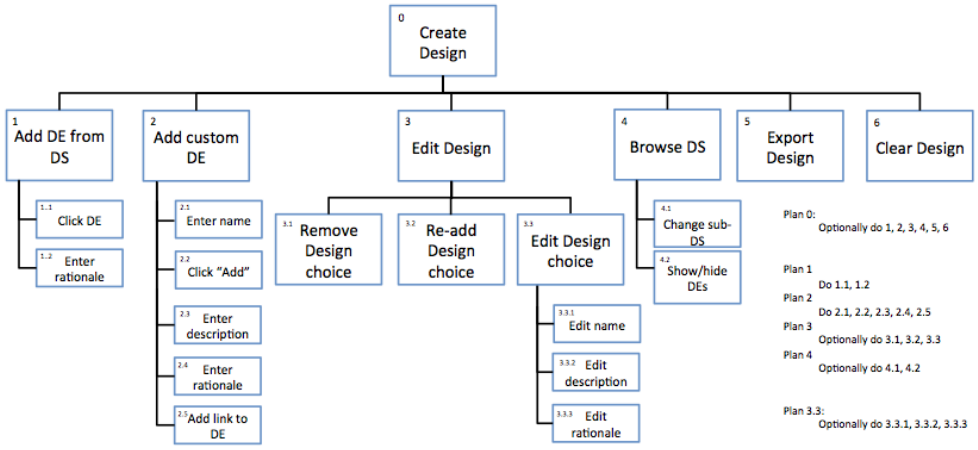


Figure 5-14: HTA for generating a design using the design space tool.

The HTA comparisons yielded one change to the design of the design space tool: ensuring that recording the stage in the design space creation process is optional. This was identified because not all methods that are used to create an explicit design space will be broken down into stages – given that prior methods found in the literature were not broken down this way (e.g. [Minhas et al., 2012, Grammel and Storey, 2010, Na et al., 2010]).

There were no necessary changes identified for either the application profiling or design generation sections of the design space tool from our Task Analyses. Since we were unable

to create a HTA for design generation without tool support, we had no base of comparison for the HTA generated the task of design generation with tool support.

We believe that using the HTAs in this way was valuable because it meant we were able to demonstrate that the design space tool is suitable to support the use of design spaces in the software engineering process. We were able to demonstrate that the tool supported design space creation and application profiling, and argue that the suitability of the design generation module is demonstrated in the empirical study in Chapter 6.

5.7.2 Cognitive Walkthrough

A cognitive walkthrough is a method for assessing the usability of a system with no user involvement. The method requires designers and evaluators "walking through" a set of typical users tasks in order to assess usability problems in the system [Wharton, 1992, Rogers et al., 2011].

Walkthroughs are an analytical evaluation method, usually considered an alternative to heuristic evaluation [Rogers et al., 2011]. Walkthroughs do not normally involve users (with the exception of pluralistic walkthroughs), meaning that the costs of evaluation are much lower than empirical methods [Rogers et al., 2011]. We decided to carry out a cognitive walkthrough because of the task-focused nature of the walkthrough.

We chose cognitive walkthrough over heuristic evaluation because we are interested in the tasks to be completed by the user rather than the more general approach featured in heuristic evaluation. A cognitive walkthrough involves the following steps [Rogers et al., 2011]:

1. Identify and record the characteristics of target users of the system, and develop a set of tasks to be assessed in the walkthrough. Describe the interface of the system. Finally, identify the actions that need to be completed using the system in order for the user to complete the specified tasks.
2. A designer of the system and one or more evaluators perform the analysis.
3. Work through each of the actions identified in the task in a typical scenario, and answer the following questions:
   (a) Will the next correct action be obvious to the user?
   (b) Will the user notice that the correct action is available to them?
   (c) Will the response from the system prompt the user to perform the next action correctly?
4. Record the following additional information: assumptions about what might cause the user problems, and why these are identified; additional thoughts on proposed changes to the design; a summary of the results from the walkthrough for that task.

There are other aspects that should be considered in each stage of the cognitive walkthrough [Wharton, 1992]:

- The environment in which the system is being used.
- The knowledge that the user is assumed to have at that stage.
- The state of the system at that stage.
- The task being completed, and in particular the complexity of the goals that the user has at that stage.
- The actions that the user needs to complete.

It is suggested that the walkthrough be carried out by filling in seven forms, in order to ensure that the walkthrough is completed comprehensively and all the relevant information is recorded throughout the walkthrough [Wharton, 1992]. The forms to be filled in specify the task to be completed, information about the user, their goals, the state of the system, and the steps required to complete the task. The rest of this section provides an overview of the salient information from the various cognitive walkthrough forms. A full report of the cognitive walkthrough is shown in Appendix E. After the cognitive walkthrough summary, we discuss the results of the walkthrough and any changes that were identified.

Task Selection We evaluated each of the three tasks that the design space tool was created to support: design space creation, application profiling and design generation. These tasks were considered in a general sense, rather than by identifying a specific design space that needed to be created, or specific tools that needed to be profiled. Each task assumed that the user had some other mechanism for generating the knowledge or information that they required to perform the task. For instance, if the user's goal was to add a design element to the design space using the tool, it was assumed that the user already knew what design element they wanted to add through some other method (such as our design space creation method specified in Section 4.3) since our evaluation is based on the support provided by the tool. For the design generation task, we decided not to suggest how the user would create the design. Instead we discuss each of the features that the tool has which support this process, and how each of these features affects the user.

User Assumptions Since our tool is meant to support the software design process, we assume that the intended user has some experience in software design for all of the tasks in the walkthrough. For design space creation and application profiling, we also assume that the designer has some knowledge of the domain, although in application profiling the domain knowledge could have been obtained in the design space creation phase.

Task Evaluation The design space creation task was broken down into seven stages:

1. Create new explicit design space
2. Edit new explicit design space
3. Create a new stage
4. Add a new source
5. Add a new design element
6. Position the design element
7. Repeat for other design elements

The application profiling task was broken down into seven stages:

1. Create a new application
2. Choose the application
3. Add profiling information for an app
4. Enter Export mode
5. Export profiling information for an app
6. Export all profiling information as heat map
7. Export all profiling information as table

The design generation task was broken down into eight stages:

1. Show design space element info
2. Choose a design space element for the design
3. Add a custom design solution
4. Clear design data
5. Alter the information being shown
6. Un-choose a design element
7. Re-choose an un-chosen design element
8. Export design

5.7.3 Evaluation Summary

One aspect that was found to be a problem throughout the cognitive walkthrough was the terminology and language that was used in the design space tool. This problem occurred because participants were not clear of the difference between the process of generating the design and creating the design space because that version of the design tool referred to both of these processes as 'creation'. Furthermore, confusion arose between the design tool and the applications in the domain as both used the term 'tool'. Thus, we now distinguish between design space creation and design generation, as well as the design space tool and EUSC applications.

The cognitive walkthrough demonstrated that the design space tool supported the action flow of both the design space creation task and the application profiling task. It also highlighted that the tool provides a number of features that are useful to design generation with design spaces, although it is not clear how a designer could use this function of the tool to help them generate new designs.

We have been unable to evaluate the task support of the design generation module as thoroughly as the other two because we did not have a HTA against which to compare. We will also demonstrate the use of the design generation module of the design space tool in the next chapter.

5.8 Limitations and Future work

The main limitation of this work is that we were not able to fully evaluate the task support of the design generation module of the design space tool since we did not have a task

specification without the tool with which to compare against the task specification in the tool.

We only performed analytical evaluation of the modules of the design space tool, rather than any empirical evaluation. We demonstrated the design space tool can be used for design space creation and application profiling in the previous chapter, as well as performing an analytical evaluation of how these tasks can be used in this chapter. We demonstrate the use of the design generation module through our exploration of the use of design spaces for design generation in the next chapter.

Immediate future work for the design space tool is to evaluate how the tool can be used in the real world by real designers. We would then be able to verify the results of our analysis and evaluate how well the design tool can support designers in their use of design spaces throughout the software engineering process.

5.9 Chapter Summary

This chapter addressed our third research goal: **RG3**. **Create and evaluate a Design Space Tool to facilitate the design and creation of design spaces and domain applications.** We identified the creation of a design space tool being an important aspect of the use of design spaces, based on our experience of creating the design space and suggestions from prior work [Baum et al., 2000].

We identified three tasks that the design space tool needed to support, based on prior work on design spaces and our experience of creating an explicit design space:

1. **Design space creation**: Creating a new design space by adding design decisions, potential design solutions and links between them.
2. **Application profiling**: Recording the design choices that are made in domain applications in the design space.
3. **Design generation**: Using the design space to generate a new design for an application in the domain.

The first part of this chapter described the creation of the design space tool, providing details of aspects such as the model that underpins the explicit design space, the requirements for the design space tool, and details of its design and implementation. We then described each of the modules of the tool, and related them to the tasks that the design space tool was designed to complete. After describing the tool, we used analytical techniques to evaluate how well the tool supported the design space tasks.

We first performed hierarchical task analysis (HTA) to determine the task breakdown of each of the three main functions of the tool. We then compared these with HTAs from information in prior literature as to how these tasks would be completed. This was successful for the design space creation and application profiling tasks, but due to a lack of specification for the

design creation task, we were unable to generate a HTA for performing design generation without the aid of the design space tool.

Following the HTA, we carried out a cognitive walkthrough with the aim to match potential users' action flow compared with what the designer of the design space tool envisioned. The cognitive walkthrough found numerous usability problems relating to terminology and feedback to the user. The cognitive walkthrough had similar issues to the HTA regarding the design generation task due to the lack of specification for this task outside of the design space tool.

The aim of this chapter was to create and evaluate a tool that would allow designers and software engineers to interact with design spaces at various points throughout the software engineering process. In the chapter, we documented the creation of a design space tool to support three tasks identified from prior work, and analytically evaluated how these tasks could be performed. We both demonstrated and analytically evaluated the tasks of design space creation and application profiling, whereas the design generation module has been analytically evaluated. This design generation module is demonstrated as part of the study reported in the next chapter.

CHAPTER 6
DESIGN GENERATION USING DESIGN SPACES

6.1 Chapter Introduction

In Chapter 2, we reviewed EUSC and design spaces, and identified how design spaces could be useful to designers. In Chapter 4, we clarified three tasks in the software engineering process that design spaces could support: design evaluation, application profiling, and design generation. Chapter 5 specified a tool that was meant to support these tasks, along with support for the creation of an explicit design space. Whilst evaluating the design space tool, we noted that there is no information as to how design spaces should be used in design generation in prior work. Hence, we decided to perform this exploration using the design space for EUSC applications that we created in Chapter 4.

This chapter presents a study designed to explore the impact that providing participants with a representation of an explicit design space can have on an initial specification of the design of a software artefact. The participants in our study used an early prototype of the Design Tool presented in Chapter 5 that only presented the design generation module to record their design choices. Following a short introduction to EUSC, participants were asked to generate a 'conceptual design' – a list of features – that they would choose if they were to design an EUSC application. Participants were provided with one of three levels of design space support: no support (no design space), partial support (design decisions only), and full support (design decisions and potential solutions). The representation of the design space was also varied across participants to ensure that it was the design space that changed the outcome of the designs, and not the single representation of the design space that was provided by the design tool.

The main aim of this chapter is to explore how design spaces can impact the design ideas generated by designers in the process of designing a software artefact. As part of this aim, we considered the design space as being able to provide three quantifiable levels of support: none, decisions only, and decisions with corresponding solutions. It is important that we

do not restrict our results to only apply to a single possible representation of design spaces (i.e. the tree hierarchy shown in multiple figures in the previous chapter), we also varied the representation of the design space between a tree and a list so that we would be able to validate the results of varying the level of design space support provided across these two different representations of the design space. Thus, we have two high-level objectives:

1. Explore the effect of design space support on designs created by designers.
2. Explore the effect of the representation of the design space on designs created by designers.

This study should also demonstrate the utility of the design space tool and how it can be used for the generation of designs in a given domain.

Given that EUSC is an ill-structured problem – one with many solutions and many ways to reach these solutions with no consensus on which solution is best – there is no objective way of measuring the difference between two designs for EUSC applications at face value. Instead, we identified a number of general properties of designs that we could compare between the designs that our participants generated. These properties were [Sinderen, 1995]: correctness, consistency, propriety, generality (separated into abstractness, specificity, and complexity) and cleanliness (how 'balanced' the design is). Furthermore, we were interested in the number of novel design choices that our participants generated to determine whether the presence of an explicit design space might impact designers' creativity.

6.2 Method

6.2.1 Design

Our aim is to explore the effect that the inclusion of an explicit design space has on the output of the early stages of design within the software engineering process. [Sommerville, 2011] states that software engineers to model applications in the domain to identify requirements – we believe design spaces are also usable in this way. We suggest that designers can use this model to create a model of the design that they want to create. This process is analogous to conceptual design in engineering [Wood and Agogino, 2005, Brunetti and Golob, 2000], but is evident in design literature as the intersection of the domain familiarisation and idea generation: where the designer uses their knowledge of the domain at that point to suggest features that they desire of the artefact to be designed.

The task that participants were asked to achieve was to generate a conceptual design (list of features, or model) of an EUSC application. They achieved this by making design choices to their design using an early prototype of our design space tool. The tool allowed them to add design choices to their design in two ways, depending on the condition of the experiment:

1. Choose a design solution from the explicit design space and record this in their design.

2. Generate and record their own custom design choice.

We will discuss the effect of the condition of the experiment on how participants could enter their design choices in the next section.

Since the task that our participants were asked to complete in the study was to produce a 'conceptual design' for a piece of software, we required that they had some experience in software deign prior to taking part in the study. Whilst it might be interesting to explore how non-designers performed with design space support, we felt that if participants did not have this experience they would find the task to be too difficult, particularly those who were not provided with any support from the explicit design space. Specifically, we required that participants had designed and implemented a piece of software based on some specification document, which could either be provided by a third party, or that they generated themselves.

The only requirement that participants' design had to fulfil was that their conceptual design described an EUSC application. Our requirements gathering study indicated that finding participants who had experience using EUSC applications was not trivial, and even with its more recent increase in popularity, it is still a niche topic. Therefore, we did not impose any requirements on participants' prior knowledge of or experience with EUSC, and as such we provided an introduction to the domain at the beginning of each session.

The study we present used a 2×2 design, plus one control group. The level of design space support and representation of the design space were manipulated independently as between subject factors.

Independent Variables

Our first independent variable is the level of support that is provided to the user by the design tool, referred to as "DS Support". This support is reflected by the amount of information that is provided to the participant using the design space. DS support has three levels:

1. **[No support]**: The participant is not provided with a design space at all, and must enter their own design choices manually.
2. **[Partial support]**: The participant is presented with a design space that only contains the design decisions that they might want to consider. Participants must enter their design choices manually.
3. **[Full support]**: The participant is presented with a full design space, containing both design decisions they might want to consider and potential solutions to these decisions that have already been identified. The participant may choose the design solutions from the explicit design space or manually enter their own design choices.

The difference in the design tool across these conditions is manifested only in the design space representation and the consequent interactions that the participant can have with that

design space. The rest of the tool (heading bar, sidebar) remains the same across each of these three levels of support.

If we were to vary the level of DS support within participants, we would need to randomise the order these levels are presented across participants to ensure that there is no learning effect. Since full support provides more information to the participant than partial support (design solutions), which in turn provides more information than no support (design decisions), participants would remember information from higher levels of DS support if they were subsequently presented with a lower level of support. Thus, DS support needed to be varied across participants.

In the implementation of the design tool, we decided to represent the explicit design space as an n-ary tree because this was an effective way of conveying the relationships between design elements. For this study, we had to ensure that we were measuring the effect of the design space support, and not specifically the effect of the tree-based representation of the design space support. Thus, we varied the representation of the design space across participants, which was referred to as "DS representation". DS representation had two levels:

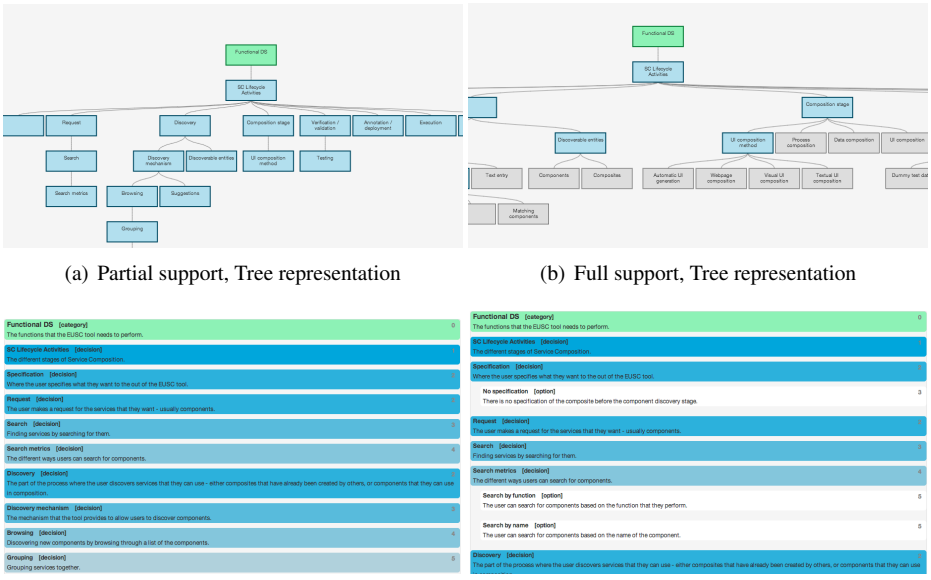
1. **[Tree-based representation]**: The decisions and options in the design space are represented as an n-ary tree.
2. **[List-based representation]**: The decisions and options in the design space are represented as a single list.

It is worth noting that since the participants who receive no support are not provided with any support from a design space, we cannot ascribe a representation to this condition of the experiment. However, we felt it necessary to include them so that we could compare the effects of the levels of DS support and DS representation against a set of participants who were carrying out the study without any assistance from a design space. There are no within-subjects IVs. Figure 6-1 shows the explicit design space in each of the four conditions of the experiment where participants were presented with a design space.

Measures

EUSC is an ill-structured problem, and as such there are a myriad of valid designs for EUSC applications making it impossible to draw any conclusions about whether one is somehow 'better' than another based on the design as a whole. Instead, we identified a number of general properties of designs that are quantifiable and can be used to identify differences between the designs [Sinderen, 1995]:

- **Correctness:** The design should not contain any errors (design solutions that are objectively wrong) [Sinderen, 1995]. This is linked with correctness of requirements, which we discuss in Section 3.6.1.
- **Consistency:** The design should be coherent, and have a clear focus [Sinderen, 1995].



(a) Partial support, Tree representation

(b) Full support, Tree representation

(c) Partial support, List representation

(d) Full support, List representation

Figure 6-1: The view of the explicit design space that users were presented with when the design space was provided

Design decisions should not contradict one another: *"do not include what is conflicting with previous design choices."* [Sinderen, 1995].

- **Propriety:** Elements in the design must be relevant to the overall purpose of the system: *"do not introduce what is immaterial"* [Sinderen, 1995].
- **Generality:** The interplay between the abstractness, specificity and complexity of the design [Wagner and Deissenboeck, 2008].
- **Cleanliness:** This is a less well-defined property that relates to the structure and balance of the design, and can be a more aesthetic quality [Sinderen, 1995].

Additionally, we wanted to explore how the use of design spaces might affect the novelty of the designs that our participants created. Thus, we also measured:

- **Number of chosen design elements** (full support only): The number of design elements that the participant selected from the DS with which they were presented, in conditions 4 and 5.
- **Number of h-novel design choices**: The number of design elements that represented concepts that were historically novel in the domain (i.e. those that have not already been identified in applications in that domain).

We also wanted to identify if using a particular level of support of DS representation imposed a greater workload onto participants. If any of the levels of either of our IVs resulted in significantly more workload than the others, it may imply that any change in are results are a consequence of this difference in workload. However, since the task that is being completed

by the participant in the condition with no support differs greatly from the conditions with full design space support. Thus, this measurement would instead be useful to determine how the participants *chose* to approach the task.

Subjective self-assessment measures are popular due to the advantage of being easy to implement and non-intrusive [Rubio et al., 2004]. The most widely used subjective assessment is the NASA Task Load Index (TLX), where participants indicate their perceived scores for particular types of workload (e.g. temporal demand, effort, etc.), and then weight the scores for these scores. We selected the NASA TLX based on the availability of software that participants could use to perform the self assessment and thus mitigating the time requirement associated with using the paper version. Furthermore, the NASA TLX is an established workload assessment tool that has been used widely in HCI.

The TLX is measured by participants indicating their perceived values for a set of measures related to workload, before ranking these measures in a pairwise fashion to see which were the most important for the task, and hence generate an overall TLX score.

Since our TLX measure does not seek to identify the workload associated with the changes in task, but instead how the changes in the task changed participants' views of the tasks. We will discuss each of the TLX sub-scores separately to identify how we believe they may be treated by participants:

- **Mental demand**: We believe that mental demand will mostly relate to the level of DS support, in that participants who receive no support might find it more difficult to generate designs than those who are provided with full support.
- **Temporal demand**: Temporal demand is more likely to be reported high when participants run out of time. We believe that this is most likely in the conditions with full DS support since participants may want to browse the whole design space, which is large and would take a substantial amount of time.
- **Physical demand**: The physical demand of the task is very low – participants are just using a computer for an hour.
- **Performance**: Perceived performance is hard to distinguish between the tasks, since this is based entirely on participants being able to guess how well they did. There is more information against which they might be able to base this decision in higher levels of support, but it is difficult to say whether this would result in higher or lower perceived performance scores.
- **Effort**: Effort is the sub-measure that we believe deviates most from what is intended in the original task. That is, participants can choose how much effort they wish to expend on the task, as opposed to being something that is inherent in the task. A difficult task might cause participants to expend a lot of effort trying to complete the task successfully, or simply give up and expend no effort because they find the task to be too hard.
- **Frustration**: Frustration is a difficult measure to interpret, given that it is difficult to

suggest what about a task might cause participants to be frustrated.

Control Variables

As we have already discussed, we needed to control certain aspects relating to the recruitment of participants that took part in our study to ensure that, as far as possible, potential confounding variables were controlled for. In our study, these were participants' prior design experience and their knowledge of and experience with EUSC. We ensured that participants had experience of creating a design for at least one piece of software, and provided participants with an introduction to EUSC such that they would all have at least a base level of experience.

We did not want to constrain the designs that participants created during the study, and in particular, to avoid all participants generating very similar designs to one another. Any other constraints we imposed might have affected the design choices that participants made, particularly those presented within the explicit design space. Thus, the only constraint that we imposed upon their designs was that they should create a conceptual design for an EUSC application, across any platform they chose, and based in any domain.

6.2.2 Hypotheses

The aim of this chapter is to explore how design spaces can be used in the design of EUSC applications, and in particular we explore the effects of changing the level of design space support provided to participants, and the representation of the design space with which they are provided. We have two overall null hypotheses, one for each of our independent variables: DS support and DS representation:

$H_0(S)$: The level of DS support provided does not affect the design participants produce.
$H_0(R)$: The DS representation does not affect the design participants produce.

These null hypotheses correspond to the following two-tailed experimental hypotheses:

$H_E(S)$: The level of support provided significantly affects the design participants produce.
$H_E(R)$: The representation of the explicit DS significantly affects the design participants produce.

These hypotheses are at a very high level, as they do not discern what types of effect that varying the level of DS support or DS representation might have. To resolve this, we broke these initial hypotheses down into more fine-grained hypotheses based on the properties that will be used in the analysis of the conceptual designs created by participants. These general properties of designs were chosen to evaluate participants' designs because of the difficulty

inherent in evaluating designs for EUSC applications, and are discussed in more detail in Section 6.2.1.

We do not have any particular reason to believe that the level of DS support or DS representation should affect the correctness, consistency or propriety of the designs participants created. However, we felt it necessary to ensure that there was not a difference in any of these properties as differences in these properties might jeopardise the other results of the study.

Each two-tailed experimental hypothesis listed below has a corresponding null hypothesis, all of which are omitted for brevity. The experimental hypotheses that will be tested in our analysis and evaluation of the designs created by participants are as follows:

$H_E(\mathbf{1})$**:** The correctness of participants' designs is affected by:
- (a) Level of DS support
- (b) DS representation

$H_E(\mathbf{2})$**:** The consistency of participants' designs is affected by:
- (a) Level of DS support
- (b) DS representation

$H_E(\mathbf{3})$**:** The propriety of participants' designs is affected by:
- (a) Level of DS support
- (b) DS representation

$H_E(\mathbf{4})$**:** The generality of participants' designs is affected by:
- (a) Level of DS support
- (b) DS representation

$H_E(\mathbf{5})$**:** The cleanliness of participants' designs is affected by:
- (a) Level of DS support
- (b) DS representation

We are interested in one other aspect of participants' design outside of these general properties: the novelty of the designs. Thus we also include the following two-tailed experimental hypothesis:

$H_E(\mathbf{6})$**:** The number of historically novel (in the domain) design choices made in participants' designs is affected by:
- (a) Level of DS support
- (b) DS representation

Thus, the goal of our study is to explore how the use of design spaces – levels of support and their representation – affect a number of general properties of the designs that participants generate. The evaluation of each of these hypothesis is reported in Section 6.3.

6.2.3 Participants

We recruited 40 participants (34 male, 6 female) across the 5 conditions of the experiment, yielding 8 per condition. Each session lasted at most 2 hours. Our participants had a mean age of 26.65, (SD = 4). 29 were students, 10 employed, and 1 self-employed. 4 had used EUSC applications before, but none reported that they use EUSC applications on a day-to-day basis. All participants had some experience in designing software, either through personal projects, work, or undergraduate coursework. Participants were recruited through advertisements placed in departments of the university where students or staff might meet our minimum requirements.

6.2.4 Apparatus and Materials

The study was carried out in a lab on a 27" iMac. A computer with a large screen was chosen to maximise the number of elements of the DS that could be shown on screen at any one time. Introductory materials were provided to participants on paper, all of which are available in Appendix F:

1. **Consent form:** A brief overview of the task, and other information relating to data protection, withdrawal from the study, etc. (See Appendix F.1).
2. **Questionnaire:** A general questionnaire used to gather information on demographics, technical knowledge, design experience, and knowledge of SC (see Appendices F.2 and F.10).
3. **Introduction to task/session:** An overview of the session, with a focus on the main task to be completed: creating a conceptual design for an EUSC application. An example conceptual design was given for a video chat application – another ill-structured domain with which the participant is assumed to be more familiar (see Appendix F.4).
4. **Introduction to SC:** An introduction to the domains of SC and EUSC, with a description of the overall aims of the process, followed by an introductory video with a number of example EUSC applications (see Appendix F.5).
5. **Task instructions:** Instructions tailored to each of the different conditions of the experiment, detailing what the participant needs to do, and how to use the different aspects of the design tool (see Appendix F.8).
6. **NASA TLX introduction:** An introduction to what the TLX is meant to achieve, and instructions stating how to complete the TLX (see Appendix F.9).

As described in Chapter 5, the design tool was implemented as a Web application; the front-end was written in HTML, CSS (Bootstrap [21]), JavaScript (jQuery[22], and JavaScript InfoVis Toolkit[23]), and the back-end was written in PHP and connected to a MySQL database, which

[21] `http://getbootstrap.com`
[22] `http://jquery.com`
[23] `http://philogb.github.io/jit/`

is where participants designs were stored anonymously. Participants' design were logged to the database continuously throughout the session. An Adobe Air implementation of the NASA TLX subjective assessment of workload was used at the end of the session to measure participants' perceived workload.

6.2.5 Procedure

Study sessions were split into three parts: an introductory phase, where participants were introduced to the session, SC, and the task to be completed, as well as completing an introductory questionnaire; the design phase where the participant creates the design for the SC application; and finally the post-design phase where participants carry out a subjective workload assessment and finally filling in another questionnaire. The design phase was limited to 60 minutes, and the rest of the session took approximately 60 minutes, but was not explicitly limited.

Session Introduction

The introductory section of study sessions had several objectives:

1. Introduce participants to the session in general, providing them with the overall aims of the session and an overview of what the study session will entail.
2. An introductory questionnaire to assess participants' prior expertise in design and EUSC, in order to ensure that they fit with our requirements for participants.
3. Introduce participants to EUSC in order to provide them with the base level of knowledge that they will require to complete the task.
4. Introduce participants to the task that they will be going to complete, and to the design tool that they will be using to complete the task, depending on the condition of the experiment that they are in.

Introduction to Session The aim of the introduction to the session was to introduce participants to the task that they would be expected to complete, and to give them an understanding of what the session as a whole would contain. They were also asked to fill in a consent form. The introduction to the session informed participants that they would need to create a 'conceptual design' for an EUSC application, and explained exactly what it is they would be expected to create. We provided an example conceptual design for a video chat application – a domain with which participants were assumed to be more familiar (compared with SC). This example conceptual design contained a number of concepts and features that a video chat application might have, split across the categories of functional, non-functional, structural, and entities – as with our earlier work on design spaces.

Participants were provided with three questionnaires to evaluate their prior experience in design, SC, as well as a general demographics questionnaire.

Introduction to Service Composition The introduction to SC was split into two parts: a general introduction to the general ideas of service composition, followed by separate introductions to a number of different EUSC applications.

The general introduction contained an overview of the underlying concepts of SC in general, as well as detailing the stages that are involved in the process. The script that was used to introduce participants to SC is provided in Appendix F.5.

The second part of the introduction to SC demonstrated tasks being completed using 7 different EUSC applications on video. Participants were asked to complete the task (or a slightly simpler version thereof) using the tools themselves. The tasks were demonstrated to participants in pre-recorded videos, which were played to participants in a random order. The order in which participants used the EUSC applications was also randomised. The tools that were demonstrated to participants were: IFTTT, Automator, Quartz Composer, Yahoo! Pipes, Atooma, Tasker and AutomateIt. The tools were selected to present participants with a wide range of designs, and the tasks that we demonstrated are listed in Appendix F.5.

When performing the tasks themselves, participants were told the task that they needed to perform using the EUSC application, and invited to do so from what they remembered from the video that they had just watched. They were informed that this was not a test, and that they could have assistance from the experimentor if they required it. Overall, the introduction to SC took approximately 30 minutes.

This introduction may prime participants to the design choices that were made in the EUSC applications with which they were presented. However, we felt that this was acceptable so long as all participants were primed to to the same set of EUSC applications. Furthermore, the EUSC applications were selected to provide as wide a range of design choices as was possible in currently available examples.

Introduction to task and the Design Space Tool The introduction to the primary experimental task differed per condition of the experiment. This introduction clarified exactly what participants needed to do in the session, as well as providing instructions as to how they could use the design tool (and associated design space) in this task.

The first part of this introduction was the same across all conditions in the experiment. First, we reiterated the output that participants were expected to create in the session: a conceptual design for an EUSC application, and clarified that they understood what it is we wanted them to achieve.

The main part of the task introduction varied based on the level of DS support provided in the condition:

- **No support/control group (Condition 1):** In this condition, participants were only provided with a mechanism for recording the design choices that they decided should be part of their conceptual design. This meant that the only instruction participants required was to be told how to use the design tool to record their design choices.
- **Partial Support (Conditions 2 and 3):** In conditions 2 and 3, participants were presented with a representation of the explicit design space that only contained the design decisions that they might want to consider. The difference between the two conditions was the representation of the explicit design space: condition 2 was tree-based, and condition 3 was a list. Thus, participants needed to be introduced to the idea of a design space, and we demonstrated how some of the design decisions that were shown were linked to one another. We instructed participants about how to record their design choices using the design tool, how they would navigate around the various decisions in the design space, and how they could find out more information about each of the design elements in the design space.
- **Full Support (Conditions 4 and 5):** In conditions 4 and 5, participants were presented with a full concrete design space that contained both design decisions that participants might want to consider, and potential solutions to solve these design decisions. Condition 4 had a tree-based DS representation, and condition 5 had a list-based DS representation. Participants first received the same instructions as those for partial support, before being instructed on how they could record the suggested solutions from the explicit DS as part of their design.

We did not place any constraints on participants in terms of how much they could view and use the EUSC applications that were demonstrated to them earlier in the session. Participants were instructed to inform the experimentor once they had finished the design task, even if they finished before before 60 minutes had elapsed.

The Design Task

In the design tasks itself, participants were asked to create the conceptual design for an EUSC application. They were not provided with any other constraints in how the EUSC application should operate, what platform(s) it should run on, or the domain that it should target.

Participants than had an hour had to generate their conceptual design by adding design choices – either their own design choices or choices selected from the design space. Once they had completed this process, we recorded the design in a database.

6.2.6 Post-Design Phase

To ensure that the perceived amount of workload in each of the conditions of the experiment remained consistent, we asked participants to complete a NASA TLX subjective workload self-assessment immediately following the completion of the design task.

The final aspect of the study was a post-questionnaire, where participants were asked to give an overview of the design that they had created, as well as being invited to give any comments they might have on the experiment, and the design tool that they used to record the design.

6.3 Results and Analysis

The designs that participants created varied widely in their size, focus on features, and the domain in which they targeted. Furthermore, the designs themselves are too large to present either in the body of this thesis, or in the appendix.

For clarity, we present an overview of two designs created by participants. The first was an EUSC application that the participant designed to support home automation (provided with no support) and the second was a more general EUSC application meant to be used on both mobile and desktop, without an explicit specification of a target domain (provided with full support).

Figure 6-2 shows an overview of the functional category of the design generated by participant 31, who was in a condition of the experiment where no DS support was provided. The functional design choices and their parent decisions are shown in the hierarchy, and the novel suggestions are shown in the side bar. Note that this representation was created once the custom design solutions that the participant suggested had been mapped to the same concept in the design space, those in the sidebar could not be mapped to elements in the design space.

We only show the functional category of the design space because presenting all four categories would take up too much space. This representation also omits the descriptions and rationale of each of the elements.

Figure 6-3 shows an overview of the design generated by participant, who was in a condition of the experiment where full DS support was provided. We show only the functional category to save space, and the participant did not identify any new design choices.

We will identify a number of topics related to the differences between these designs later in the analysis.

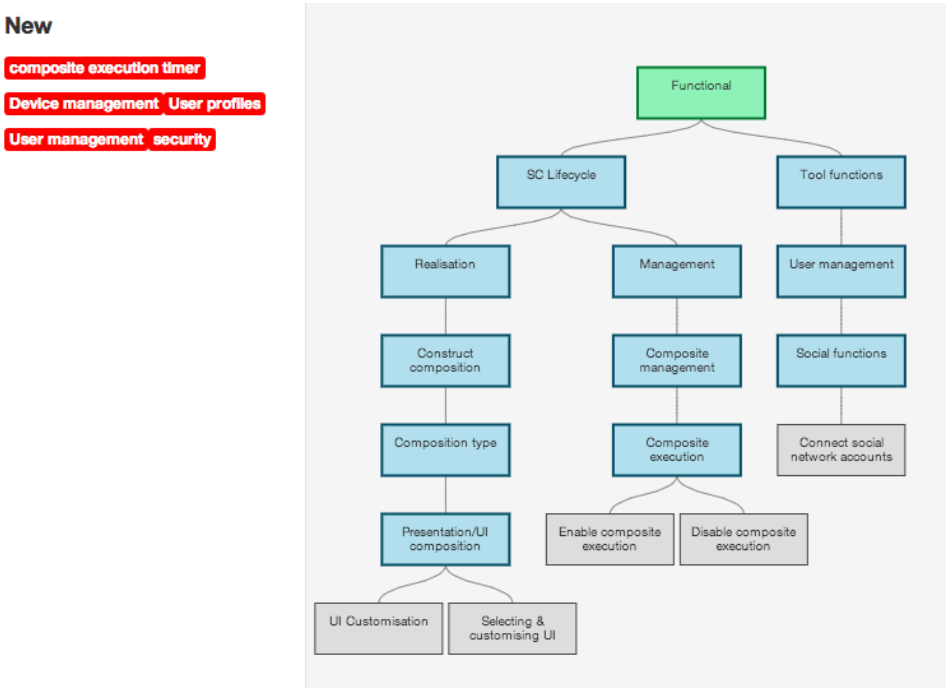


Figure 6-2: An overview of the design generated by participant 31 (showing functional and custom design choices only).

6.3.1 Analysis

The design tool recorded two different types of design solution within participants' conceptual designs: solutions chosen from within the explicit design space, and choices suggested by participants.

Design solutions in the explicit design space were all at the same level of granularity because of the method used to create it, i.e. each solution represented only a single design choice that could be made.

The same cannot be said of the custom solutions, since participants were not given any guidance as to the level of abstraction at which their custom solutions should be. This meant that in order for us to be able to compare the custom design solutions with the ones from the design space, we first needed to break the custom solutions down so that they reflected a single design choice, and were at the same level of granularity as the solutions in the explicit design space.

We split the custom design solutions based on the contents of their description and rationale entered by the participant. We identified the main topics within the description and rationale, and generated a new design solution for each of the separately identified topics. These split custom design choices were then directly comparable with those already in the explicit design space. Table 6.1 shows an example of splitting design choices where participant 2

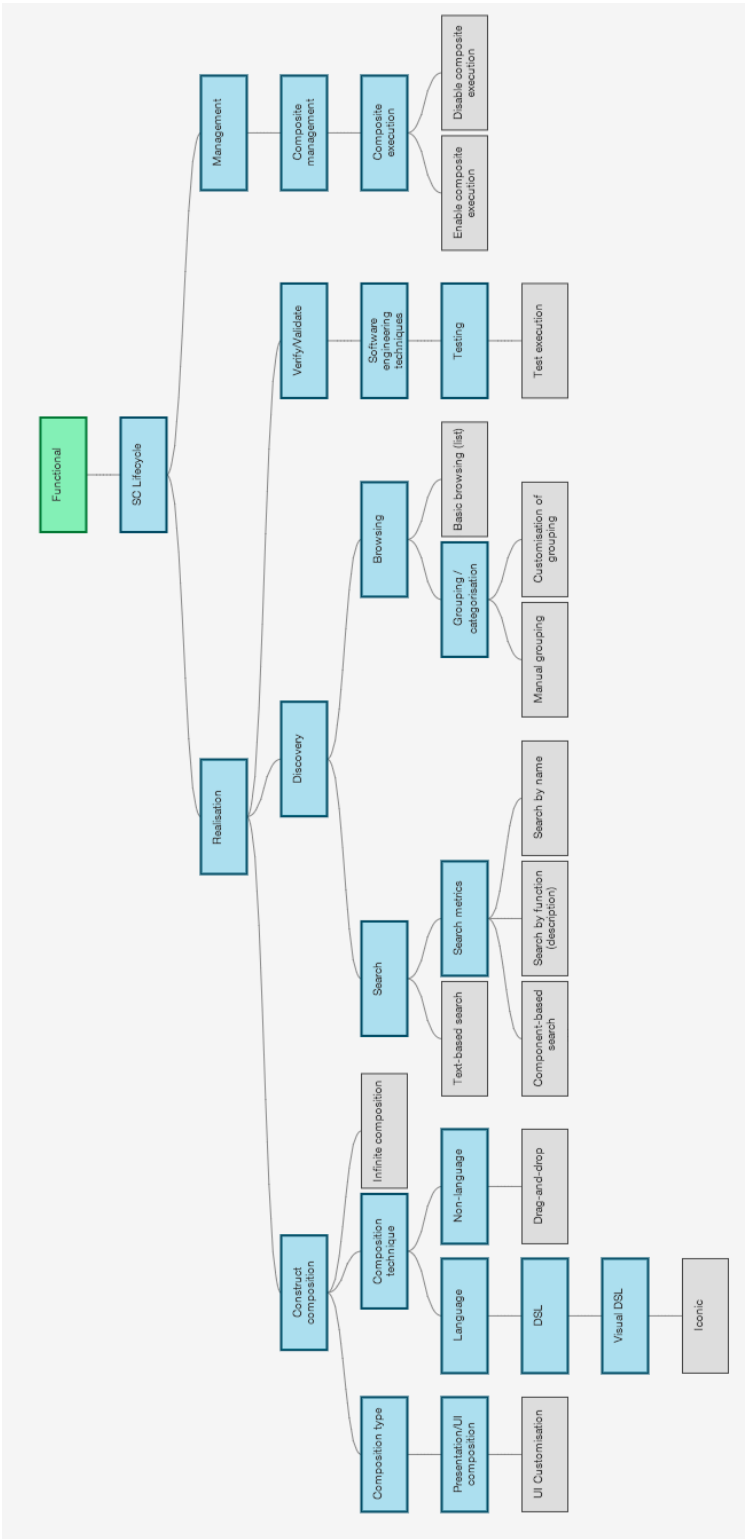


Figure 6-3: An overview of the design generated by participant 37 (showing functional design choices only).

suggested that their EUSC application would include some form of testing of composites as part of its functionality.

Table 6.1: Example of the design element splitting process.

Original Design Choice (provided by participant)	
Name:	Test Composition (16)
Description:	Dry-run of the composition. It may be necessary to provide mechanisms for providing dummy data – for example, it is not desirable to have to drive a car at 20mph to ensure that the composed rule works correctly – should stub out the data source so that you can tell the composition that the GPS is moving at 20mph.
Rationale:	Ensure that the composition performs the desired function.
Split Choice 1	
Name:	Test
Description:	Dry-run of the composition.
Rationale:	Ensure that the composition performs the desired function
Split Choice 2	
Name:	'Dummy' test data
Description:	It may be necessary to provide mechanisms for providing dummy data – for example, it is not desirable to have to drive a car at 20mph to ensure that the composed rule works correctly – should stub out the data source so that you can tell the composition that the GPS is moving at 20mph.
Rationale:	Ensure that the composition performs the desired function.

Splitting up the custom design choices did not change the content of the design choices, and as such did not have an effect on the design that the participant created. The only effect of this process is to ensure that any quantitative comparisons that we make in this work are valid.

We were also interested in the number of custom design solutions that we had not already identified as being part of our explicit design space for EUSC applications, i.e. those that were novel to the domain. We processed the split design choices to identify those which were already in our explicit design space, and those which were not.

6.3.2 Design Results

Our analysis of the results of this study are split across the properties that we identified as being useful for evaluating designs in Section 6.2.1, as well as an assessment of the novelty that can be associated with design elements chosen by participants. Finally, we analyse the results of a workload assessment to identify whether any of the conditions of the study had a significantly higher or lower perceived workload than the others.

Our first analysis is based on the general properties of designs we identified in Section 6.2 correctness, consistency, propriety, generality, and cleanliness. We be performed statistical

tests of significance to measure these properties, and for these tests we assume that they meet the assumptions of normality and homogeneous variance required to perform parametric statistics. Where our data do not meet these assumptions, we will explain how the assumptions for the test were assessed and/or violated, before performing the alternative test.

A discussion of all of the significant results of the study is provided in Section 6.4.

Correctness

Correctness refers to the validity of the design choices that are made within a particular design [Sinderen, 1995]. Rather than referring to whether the participant could have made a better design choice, a correct design choice is simply one that could not be considered as being incorrect for an artefact in the chosen domain. Incorrect design choices in our participants' designs are therefore any that are not applicable to any kind of EUSC application.

To assess whether a participant's design as a whole could be considered as being correct, we evaluated whether each of the custom design choices that they made was correct or incorrect (design choices chosen from the explicit DS are correct by definition).

After evaluating each of the custom design choices made by participants, we did not find any to be incorrect, thus we can report that all of the designs created in the study can be considered as being correct. Thus, we must accept our null hypotheses for correctness, and reject the corresponding experimental hypotheses: $H_E(\mathbf{1})(\mathbf{a})$ and $H_E(\mathbf{1})(\mathbf{b})$, and conclude that neither the level of DS support nor the DS representation affected the correctness of the designs that participants created.

Consistency

As with correctness, consistency is a binary classification – a design is either consistent, or it is not. We measured the consistency of participants' designs in the same way that our requirements were evaluated for their consistency in Chapter 3. That is, we evaluated every distinct pair of design choices within a design to identify any conflicts between design choices made by participants. Both custom and chosen design choices were assessed in this way.

Conflicting design choices were identified in the designs generated by 3 participants. The conflicting design choices are shown in Table 6.2. All cases of inconsistency that we identified were when one design choice directly contradicts with at least one other, and as such are organised into pairs.

This meant that 37 of the designs generated in the study were consistent. Since there were only 3 that were not consistent, we are unable to perform any robust statistical analysis of the distribution of inconsistent design choices across conditions of the experiment. The

Table 6.2: Number of contradicting design choices per participant.

Condition	# conflicting pairs	Conflicts
4	1	High abstraction vs low abstraction
4	4	Textual DSL vs Visual DSL
		Implicit control flow vs Explicit control flow
		Public online section vs private online section
		Explicit data flow vs Implicit data flow
4	1	Full language interaction vs drag-and-drop interaction

small number of inconsistent results in this case means we should reject both $H_E(\mathbf{2})(\mathbf{a})$ and $H_E(\mathbf{2})(\mathbf{b})$ and accept the corresponding null hypotheses to conclude that the consistency of participants designs is not affected by either level of DS support or DS representation.

Propriety

The propriety of the design seeks to assess the appropriateness and relevance of the design as a whole, and as with correctness and consistency, requires the evaluation of each of the design choices within each design. Also, like correctness it was only the custom design choices that needed to be assessed for appropriateness and relevance, since all design choices that we had already placed in the explicit design space had to be appropriate and relevant by definition.

Evaluating each of the custom design choices individually indicated that no design choices were either inappropriate or irrelevant, meaning that we could conclude that all designs generated as part of the study are both relevant and appropriate. Thus, we must accept our null hypotheses for propriety and reject the corresponding experimental hypotheses: $H_E(\mathbf{3})(\mathbf{a})$ and $H_E(\mathbf{3})(\mathbf{b})$, and conclude that neither the level of DS support nor the DS representation affected the propriety of the designs that participants created.

Generality

Generality is a much more complex property to use to evaluate the designs than the previous properties we have used, since it is not a binary relation that can simply be applied to each of the design choices that were made in each design. Simply put, generality refers to how abstract the design is [Sinderen, 1995].

[Wagner and Deissenboeck, 2008] suggest that determining the abstractness of a design or software artefact is closely related to its specificity and complexity. They define the following concepts:

- **Abstractness**: The degree of information loss an argument has, i.e. the amount of detail that has been removed in order to provide a model.
- **Specificity**: The number of contexts in which the design/artefact can be used.
- **Complexity**: The interplay of two types of complexity – *detail complexity*, the number of design elements within the design; and *dynamic complexity*, the number of causal relationships between the design elements.

Since we are breaking down our analysis of generality into its constituent parts, we must also break down our experimental hypotheses $H_E(\mathbf{4})(\mathbf{a})$ and $H_E(\mathbf{4})(\mathbf{b})$. Each of the experimental hypotheses listed below has a corresponding null hypothesis:

$H_E(\mathbf{4})$**:** The generality of participants' designs is affected by the level of DS support or the DS representation

$H_E(\mathbf{4.1})$**:** The abstractness of participants' designs is affected by:

(a) Level of DS support

(b) DS representation

$H_E(\mathbf{4.2})$**:** The specificity of participants' designs is affected by:

(a) Level of DS support

(b) DS representation

$H_E(\mathbf{4.3})$**:** The complexity of participants' designs is affected by:

(a) Level of DS support

(b) DS representation

By accepting or rejecting each of the experimental hypotheses, we will be able to address the overall experimental hypotheses for generality $H_E(\mathbf{4})(\mathbf{a})$ and $H_E(\mathbf{4})(\mathbf{b})$.

Abstractness When we consider the abstractness of the design, we are considering the design as a whole that is generated by each participant, rather than the individual elements therein. In software design, abstraction should not be considered for a single element, but instead as a relationship between two elements [Krueger, 1992]. Thus, in this analysis we do not suggest how abstract a given design is, only the relative abstractness of the designs with respect to one another.

To assess the relative abstractness of the designs created by participants, we compared every design with every other design, and recorded whether it was more abstract, less abstract, or if there was no discernible difference between the abstractness of either of the designs. The output of this abstractness comparison was processed to create a graph, where the nodes of the graph represented the designs created by participants, and the edges represented an abstractness relation between these two designs. This resulted in a graph where every design was linked to every other that was considered as being more abstract, and every other design that was considered as being less abstract, and as such contained a large number of superfluous links. Links were considered as being superfluous if they 'hid' a more complex relationship. For instance, if $A \xrightarrow{abs} B$ and $B \xrightarrow{abs} C$ then we know that $A \xrightarrow{abs} C$, and the

link between nodes A and C becomes redundant. This example is illustrated in Figure 6-4.

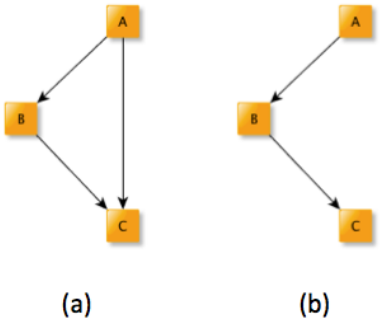


Figure 6-4: A cluster within the abstractness relation graph (a) pre-removal of superfluous links, and (b) post-removal.

We sanitised the abstractness relation graph to remove superfluous links between nodes, and processed the result using yEd[24] to organise the abstractness relations into a hierarchical structure. The result of this hierarchical structuring are shown in Figure 6-5, where each node is the design created by a participant, and a directed edge from **A** to **B** indicates that the design represented by **B** is *more* abstract than the design represented by **A**.

Figure 6-5 shows the organisation of the designs into a number of levels, which we coded as level 1 being the most abstract, and level 10 being the least abstract (or most concrete). All designs on level i are more abstract (less concrete) than those on level $i + 1$, and less abstract (more concrete) than those on level $i - 1$.

We will first attempt to resolve $H_E(\mathbf{4.1})(\mathbf{a})$ and test whether the level of support provided affected the abstractness of the designs created by participants. The level of support variable is ordinal, and has three levels: no support (condition 1); partial support (conditions 2 and 3); full support (conditions 4 and 5). We tested for correlation between abstractness level and level of DS support to identify if an increase in DS support resulted in an increase or decrease in abstractness. We calculated Spearman's rank correlation coefficient to test for monotonicity (the data are unsuitable for Pearson's product moment correlation due to the level of support being an ordinal scale). We tested each of the different representations of the design space separately to identify if the correlation differed across the representation: tree-based representation – $rs[22] = 0.702$, $p < 0.01$; list-based representation $rs[22] = 0.669$, $p < 0.01$.

We identified strong positive correlation in both groupings, indicating that the greater the level of DS support provided to our participants, the more concrete the designs they created were likely to be. There appears to be little difference in the monotonicity of the abstractness across the two representations of the DS, but to be sure of there being no difference, we need

[24] `http://yed.yworks.com/`

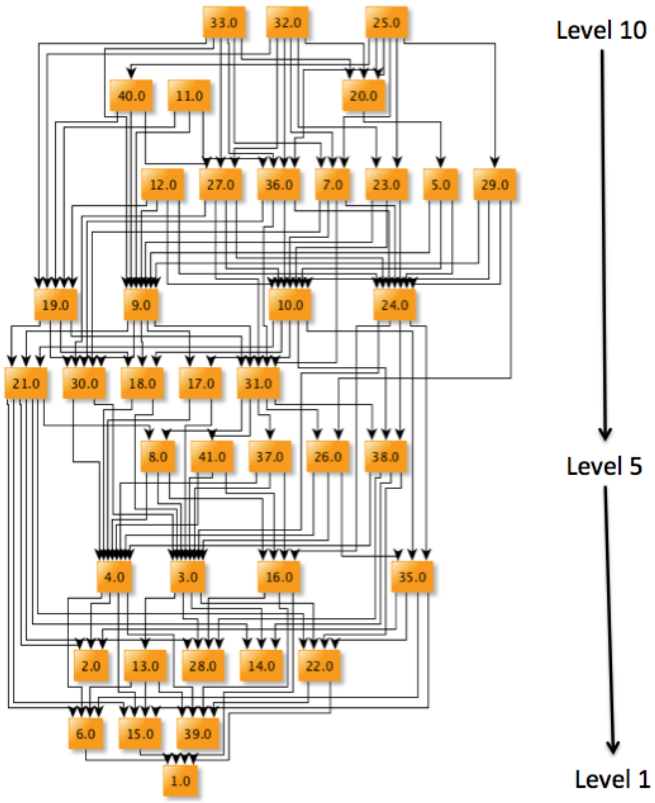


Figure 6-5: Hierarchically-ordered abstractness relations.

to test for significant difference in abstractness across the representations.

We performed a Shapiro-Wilks test for normality on the abstractness results in each condition, and found that all were normally distributed except condition 5 ($p = 0.015$). Thus, our data was unsuitable for an ANOVA or other parametric statistical tests.

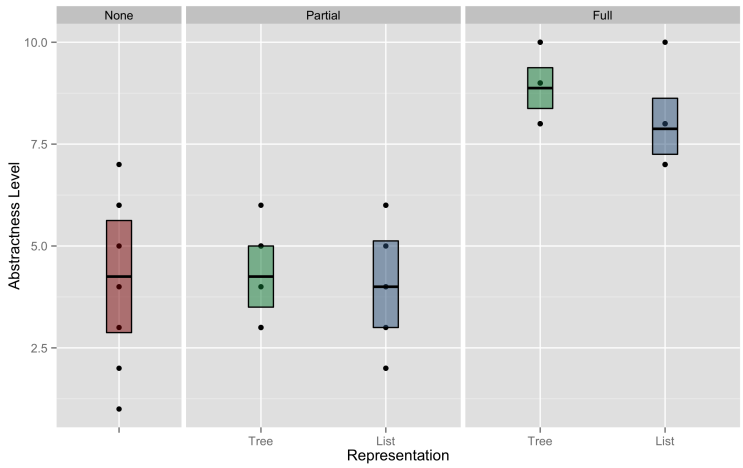


Figure 6-6: Abstractness levels across conditions (R = 1000, 95% c.i.)

Figure 6-6 shows the results of mean non-parametric bootstrapping (R = 1000, 95% confidence interval [c.i.]) of the abstractness levels across the 5 different conditions of the experiment, split by DS representation, and grouped by the level of DS support. Significant difference is identified when bars do not overlap.

It is clear from Figure 6-6 that there is a significant difference in abstractness level between each of the representations in full support (conditions 4 and 5) and each of the representations in no support (condition 1) and partial support (conditions 2 and 3). However, there is no significant difference in abstractness levels across the representations within the same level of DS support. Thus, we reject our null hypothesis for abstractness and level of DS support and accept the corresponding experimental hypothesis $H_E(\mathbf{4.1})(\mathbf{a})$ to conclude that the level of DS support had a significant effect on the abstractness of the designs that were created. In particular, the more DS support provided, the more concrete the design created. We must accept the null hypothesis for abstractness and DS representation and reject the corresponding experimental hypothesis $H_E(\mathbf{4.1})(\mathbf{b})$ and conclude that the DS representation did not have an effect on the abstractness of designs.

Specificity Measuring the specificity of a design places it somewhere on a continuum from *specific* to *generic*, and effectively refers to the number of contexts in which that particular artefact can operate [Wagner and Deissenboeck, 2008]. Our design task asked participants to design an EUSC application (without providing them with a specification of the domain in which this EUSC application should operate). Some participants had very specific ideas

about designing an EUSC application for a particular field (e.g. Computer Algebra – P8, home automation – P31, brainstorming – P21), whereas others sought to be more generic, restricting the domain based on the EUSC application's context of use (desktop vs mobile vs Web).

We applied the comparison process used to identify the abstractness levels to the specificity of the designs. We performed this step because the process of positioning the design on the specific-generic continuum, and as such the tipping point between specific and generic, is not clear. For specificity, we generated a graph that was split into 11 layers. The graph hierarchy is presented in Appendix G.

We calculated Spearman's rank correlation coefficient to identify any correlation between the DS support provided and the specificity level of the design. The correlation between the level of DS support and the specificity level for each representation are: tree-representation – $rs[22] = 0.272$, $p = 0.199$; list-representation – $rs[22] = 0.335$, $p = 0.109$. The Spearman's ρ results indicate a weak positive correlation between the level of DS support provided and the specificity of the design that is created across both representations, as well as a similar degree of correlation across representations.

A Shapiro-Wilks test indicated that the specificity levels were not normal for conditions 4 ($p = 0.020$) and 5 ($p = 0.049$), hence we performed non-parametric bootstrapping, the results of which are shown in Figure 6-7.

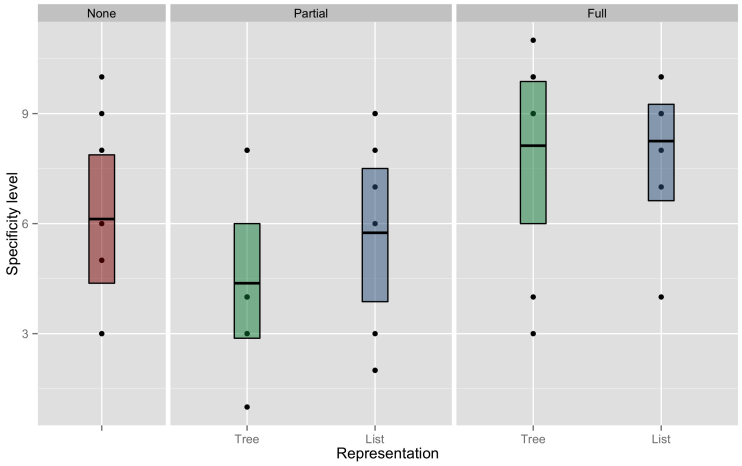


Figure 6-7: Specificity levels across conditions (R = 1000, 95% c.i.)

Figure 6-7 shows the results of mean non-parametric bootstrapping (R = 1000, 95% c.i.) of the specificity levels across the 5 different conditions of the experiment, split by DS representation, and grouped by the level of DS support. There is significant difference between both representations at full support (conditions 4 and 5) and partial support with a tree representation, but no significant difference between representations at the same level of DS support. Thus, we reject our null hypothesis for specificity and level of DS support

and accept the corresponding experimental hypothesis $H_E(\mathbf{4.2})(\mathbf{a})$ to conclude that the level of DS support had a significant effect on the specificity of the designs that were created. In particular, the greater the level of DS support provided, the more specific the design created by participants. We accept the null hypothesis for specificity and DS representation, reject the corresponding experimental hypothesis $H_E(\mathbf{4.2})(\mathbf{b})$ and conclude that the representation of the design space did not have an effect on the specificity of designs.

Complexity Our analysis of the complexity is broken into two parts based on the work of [Wagner and Deissenboeck, 2008], who suggest that the complexity of a design can be measured in terms of the number of design choices contained within it (detail complexity), and the number of causal relationships between the design choices within the design (dynamic complexity). Our first analysis of the complexity of designs is aimed at detail complexity, before moving onto dynamic complexity. We use the results of these analyses to resolve the complexity-based hypotheses $H_E(\mathbf{4.3})(\mathbf{a})$ and $H_E(\mathbf{4.3})(\mathbf{b})$ before providing a resolution to our overall hypotheses for generality.

We calculated Spearman's rank correlation coefficient between the number of design choices and the level of DS support for both DS representations (detail complexity): tree-based representation – $rs[22] = 0.845$, $p < 0.001$; list-based representation – $rs[22] = 0.616$, $p < 0.01$. This shows strong positive correlation between the number of design choices and the level of DS support across both representations, with slightly stronger correlation in the tree-based DS representation.

A Shapiro-Wilks test indicated that the complexity totals were not normally distributed in condition 1 $(p < 0.001)$, but were normally distributed in all other conditions. Thus, we performed non-parametric bootstrapping, the results of which are shown in Figure 6-8.

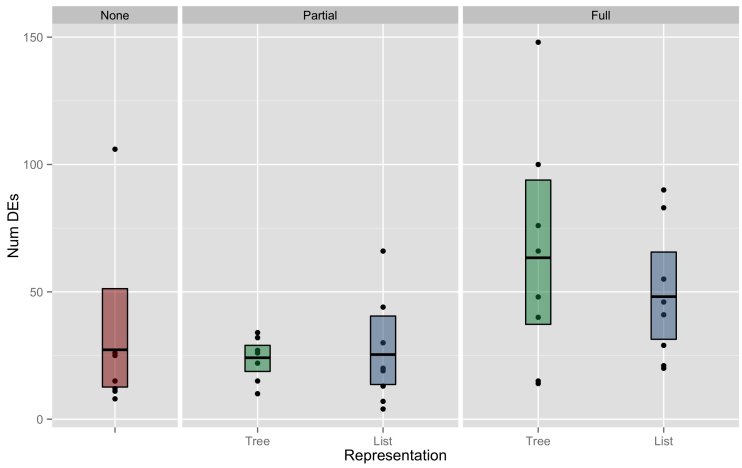


Figure 6-8: Complexity – Number of design choices across conditions (R = 1000, 95% c.i.)

Figure 6-8 shows the results of mean non-parametric bootstrapping (R = 1000, 95% c.i.) of

the number of design choices per design across the 5 different conditions of the experiment, split by DS representation, and grouped by the level of DS support. In Figure 6-8 we can see significant difference between full support representations (conditions 4 and 5) and partial support tree-based representation (condition 2). There is no significant difference in the number of design choices per design across different representations within the same level of DS support.

The second aspect of complexity is the number of causal relationships between design choices in the design. That is, for each element in the design, could it be said that choosing design element α 'caused' the participant to also choose design element β.

To identify causal relationships between design choices, we performed pairwise comparisons between each of the design choices that were chosen as being part of a design for a given participant. For each pair of design choices α and β, we sought to identify if any of the following were true:

- α 'causes' β (e.g. End-user as target user $\xrightarrow{causes}$ visual composition process)
- β 'causes' α (the opposite of the above)
- α and β could cause one another (e.g. EUSC app is Web app $\xleftrightarrow{causes}$ Website)

We analysed each participant's design and recorded the number of causal relationships within each design. The analysis yielded a total of 1803 causal relationships across the 40 designs. We then assigned each design with the number of causal relationships between design choices that were identified within it.

Spearman's rank correlation calculated between the number of causal relationships per design and the level of DS support yielded the following results: tree-based representation – $rs[22] = 0.495$, $p = 0.014$; list-based representation – $rs[22] = 0.321$, $p = 0.126$). This shows a weak to moderate positive correlation in each of the separate representations, where the list-based representation has a slightly weaker correlation than the tree.

A Shapiro-Wilks test indicated that the number of causal relationships per design were not normal for conditions 1 ($p < 0.001$) and 3 ($p < 0.001$) and hence unsuitable for assessment with an ANOVA.

Figure 6-9 shows the results of mean non-parametric bootstrapping (R = 1000, 95% c.i.) of the number of causal relations between design choices per design across the 5 different conditions of the experiment, split by DS representation, and grouped by the level of DS support. It is evident that there is significant difference between the number of causal relations per design across the support levels with the tree-based DS representation (conditions 2 and 4). There is no significant difference in the number of causal relations per design across different representations within the same level of DS support.

Figures 6-8 and 6-9 both show significant difference between partial and full DS support in the tree-based DS representation. To identify whether this difference manifested when we

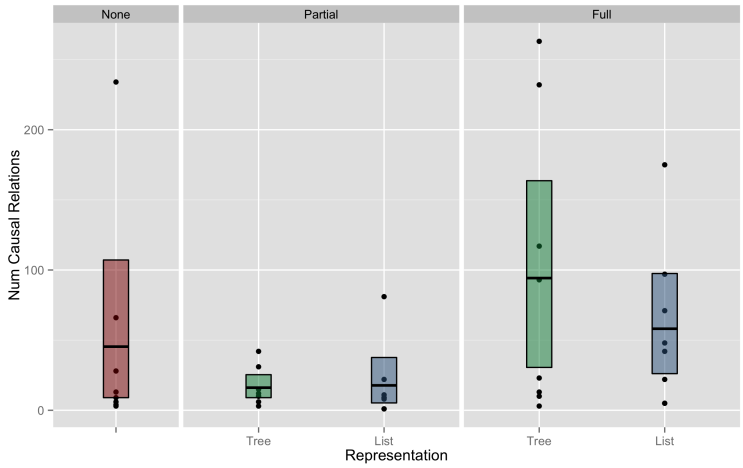
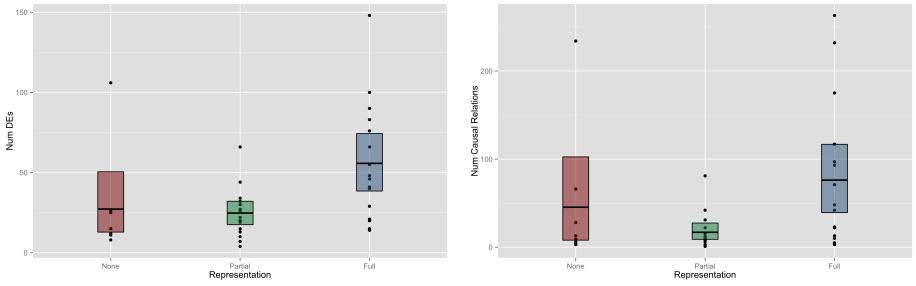


Figure 6-9: Complexity – Number of causality relations across conditions (R = 1000, 95% c.i.)



(a) Num Design Choices

(b) Num Causal Relations

Figure 6-10: Complexity across levels of DS Support (R = 1000, 95% c.i.)

remain agnostic to the DS representation, we performed mean non-parametric bootstrapping (R = 1000 95% c.i.) across levels of DS support, without considering the DS representation. The results of this bootstrapping are shown in Figure 6-10.

Figure 6-10 (a) and (b) show that when we consider the complexity whilst remaining agnostic to the DS representation, complexity is significantly different between partial and full support. Thus, we can reject our null hypothesis for complexity and level of DS support and accept the corresponding experimental hypothesis $H_E(\mathbf{4.3})(\mathbf{a})$ to conclude that the level of DS support had a significant effect on the complexity of the designs that were created. In particular, the greater the level of DS support provided, the more complex the design created by participants. We must accept the null hypothesis for abstractness and DS representation, reject the corresponding experimental hypothesis $H_E(\mathbf{4.3})(\mathbf{b})$ and conclude that the DS representation did not have an effect on the complexity of designs.

Generality Overview We accepted all three null hypotheses for DS representation relating to each of the sub-components of generality and rejected the corresponding experimental hypotheses: $H_E(\mathbf{4.1})(\mathbf{b})$, $H_E(\mathbf{4.2})(\mathbf{b})$ and $H_E(\mathbf{4.3})(\mathbf{b})$, which provides us with sufficient support to accept our overall null hypothesis, reject $H_E(\mathbf{4})(\mathbf{b})$ and conclude that the DS representation did not have an effect on the generality of designs created by participants in our study.

Conversely, we rejected all three null hypotheses for level of DS support relating to each of the sub-components of generality, accepted each of $H_E(\mathbf{4.1})(\mathbf{a})$, $H_E(\mathbf{4.2})(\mathbf{a})$ and $H_E(\mathbf{4.3})(\mathbf{a})$, which provides us with sufficient support to reject our overall null hypothesis, accept $H_E(\mathbf{4})(\mathbf{a})$ and conclude that the level of DS support did have an effect on the generality of designs created by participants in our study.

Cleanliness

There are no established methods to measure the cleanliness or balance of the designs, meaning that we have to devise our own method for evaluating this. We decided to first assess the balance of the designs across the four main categories of the design space, since the data were already grouped into these categories and they provide topics across which design choices could be made. Due to the high-level of abstraction of these categories, we performed a second analysis based on a set of sub-categories identified in prior work. Note that these sub-categories were identified *independently* from the explicit design space creation method, since evaluating the balance of the designs using EUSC-related topics that some participants were presented with would be somewhat self-fulfilling.

No prior work has provided any methods for being able to assess the cleanliness or balance of a given design, presumably due to the subjective nature of the measure. Hence, we perform a bespoke analysis of the cleanliness of the designs created in our study, with a particular focus on the equal balance of design choices across groups (either categories or sub-categories), the total number of design choices in the design (discussed above in the assessment of the complexity of the designs), and the interplay between these two measures. We believe that this balance is important because a balanced design shows the designer considered design decisions across important categories in the domain, as well as other important topics in the design of software artefacts in general.

We chose to use the four design space categories because they provide a high-level set of groups for all of the design elements, and all participants were exposed to these groups in the introduction. However, there is some priming since participants in conditions 2-5 were presented with these categories in the DS representation, whereas those in condition 1 were not. To mitigate this, we sought to identify a number of smaller categories that could break the space down into finer-grained groups, identified independently of the work on the explicit design space.

Our sub-categories were based solely on domain literature and other related work (note that the sub-categories are split into those that can be considered as functional, and those that are non-functional):

- **Functional.** These sub-categories were based on the dynamic SC life cycle [da Silva et al., 2008]: Component management, Specification, Discovery, Composition, Composite management, Execution.
- **Non-Functional.** These sub-categories were based on an extension of ISO/IEC 25010: Usability, Safety, Flexibility, Quality, Operability, Functional suitability, Reliability, Security, Compatibility, Maintainability, Transferability, Community/online, Target user, UI/representation, composition architecture, application architecture.

Our analysis uses a χ^2 goodness of fit test to compare the distribution of each of the designs against a flat distribution where the same number of design choices were made in each group (both for categories and sub-categories). Our method must account for balance across groups whilst also taking into account the total quantity. For example, a person might have only 4 design choices distributed evenly across the 4 categories [1F, 1NF, 1S, 1E], and another might have 40 design choices distributed slightly unevenly [11F, 12NF, 8S, 9E]. It is not clear which of these should have the best cleanliness score.

Below, we describe the method that was used to generate the balance scores for each of the designs, whilst working through an example distribution of design choices. The example is the distribution of design choices by participant 1: 1 functional, 0 non-functional, 5 structural, 5 entity, which is represented at [1F, 0NF, 5S, 5E]. The method for deriving the weighted scores involved the following steps:

1. Vector normalise the number of design choices per group so that (divide through by the sum value of the values): $[0.091, 0, 0.455, 0.455]$.
2. Perform a χ^2 goodness of fit test to test these normalised scores against a normalised flat distribution: $\chi^2([0.091, 0, 0.455, 0.455], [0.25, 0.25, 0.25, 0.25]) = 0.49$. This yielded a balance score for each design that is independent of the size of the design that the participant created.
3. Incorporate the number of design choices in the design w.r.t. the total number of design choices generated across all participants, by multiplying the χ^2 result by the total number of design choices in that design, and subsequently dividing through by the total number of design choices across all designs: $0.49 \times 11/1498 = 0.003$.
4. Normalise the relative χ^2 scores by dividing through by the largest relative weight : $0.003/0.05 = 0.07$. This does not affect the result of the scores, it solely normalises the values to be in a more sensible range.

After applying this method to all designs, we have the following for each design (e.g. P1[1,0,5,5]):

1. Total number of design choices in the design (e.g. 11).

2. χ^2 balance score for the design (e.g. 0.49)
3. A weighted score attempting to measure the combined effect of the total number of design choices in the design, and its overall balance (e.g. 0.003).

Since our discussion of complexity already presents a discussion of the relationship between the level of support provided and the number of design choices within a design, it will not be repeated here. For each of the other two scores, we can calculate Spearman's rank correlation coefficient for the score against the level of support provided, across each of the three groupings of our data. This calculation should provide an indication as to whether increasing the level of DS support increases or decreases the balance of the design that is generated. The results of these correlation calculations are shown in Table 6.3, along with the results of the same calculations performed against the set of sub-categories that are listed above.

Table 6.3: Spearman's ρ for χ^2 tests against balanced categories and sub-categories.

Representation	**Categories**		**Sub-categories**	
	Normalised	Weighted	Normalised	Weighted
Tree	$rs[22] = 0.56$ (p $= 0.004$)	$rs[22] = 0.833$ (p < 0.001)	$rs[22] = 0.701$ (p < 0.001)	$rs[22] = 0.833$ (p < 0.001)
List	$rs[22] = 0.387$ (p $= 0.062$)	$rs[22] = 0.612$ (p $= 0.001$)	$rs[22] = 0.562$ (p $= 0.004$)	$rs[22] = 0.623$ (p < 0.001)

Table 6.3 shows Spearman's ρ scores correlating the level of DS support against the χ^2 scores for each of the two DS representations. It is split first into categories and sub-categories, and secondly into normalised (agnostic to the size of the design) and weighted (accounting for the size of the design).

The categorical split shows moderate positive correlation for the normalised score for the tree-based representation, which becomes strong positive correlation when we also take the size of the design into account. This increase is also reflected in the list-based representation, but the normalised score starts as weak positive correlation, moving to moderate positive correlation. This indicates that the greater the support provided, the more balanced the designs across categories, and this is much more so if we also take the size of the design that participants created.

The sub-categorical split yields similar results, where strong positive correlation becomes stronger positive correlation when we consider the size of the design for the tree-based representation, and moderate positive correlation becomes moderate-strong correlation for the list-based representation.

The ρ-values for the weighted scores are very similar (Tree difference = 0, List difference = 0.01), which we believe indicates that the weighted scores are dominated by the size of participants' designs. That is, when the size of the design is taken into account, the scores reflect this size rather than its balance. However, since the normalised balance correlations are moderate-positive for the level of DS support against the χ^2 balance scores for both representations of the DS. The difference between the ρ-values is similar between the two representations across all both categories and sub-categories, and normalised vs weighted

scores.

Thus, we reject both of the null hypotheses for cleanliness and accept the corresponding experimental hypotheses $H_E(\mathbf{5})(\mathbf{a})$ and $H_E(\mathbf{5})(\mathbf{b})$ to conclude that both the level of DS support and the DS representation affect the balance of the designs created by participants. The implications of these findings are discussed in Section 6.4.

Historically Novel Design Solutions

We were also interested in the number of design choices that are historically novel (h-novel) in the domain of EUSC. That is, these are design choices suggested by participants that have not been identified either as part of EUSC applications currently available, or in research in the domain. To identify novel design choices, we reviewed each of the custom design choices entered by participants (those chosen from the explicit design space could not be h-novel by definition), and recorded whether the design choice was visible in either EUSC research, or in currently available EUSC applications. Design choices that were not recorded as falling within either of these categories were identified as being h-novel.

As with our previous analyses, we first calculated Spearman's rank correlation coefficient for both representations of the DS: tree-based representation – $rs[22] = -0.522$, $p = 0.009$; list-based representation – $rs[22] = -0.483$, $p = 0.017$. This shows that there is moderate negative correlation between the level of DS support provided and the number of h-novel design solutions that were identified by participants. In other words, participants were more likely to generate design ideas that were historically novel in the domain in conditions with lower DS support.

A Shapiro-Wilks test for normality of the number of h-novel design solution indicated that the data are not normal $(p < 0.001)$, and thus unsuitable for parametric statistical tests of difference.

Figure 6-11 shows the results of mean non-parametric bootstrapping (R = 1000, 95% c.i.) of the number of novel design choices across the 5 different conditions of the experiment, split by DS representation, and grouped by the level of DS support. This figure indicates that there is significant difference between the number of historically novel design choices per design between condition 2 and each of conditions 4 and 5 (those with full DS support). As with complexity, we performed non-parametric bootstrapping whilst remaining agnostic to the DS representation, the results of which are shown in Figure 6-12.

Figure 6-12 shows that when we consider the number of historically novel design choices per design whilst remaining agnostic to the DS representation, significantly more design ideas are generated with partial DS support than full DS support. Thus, we reject the null hypothesis for novelty and DS support, accept $H_E(\mathbf{6})(\mathbf{a})$ and conclude that the DS support had an effect on the number of historically novel design choices that participants generated. In particular,

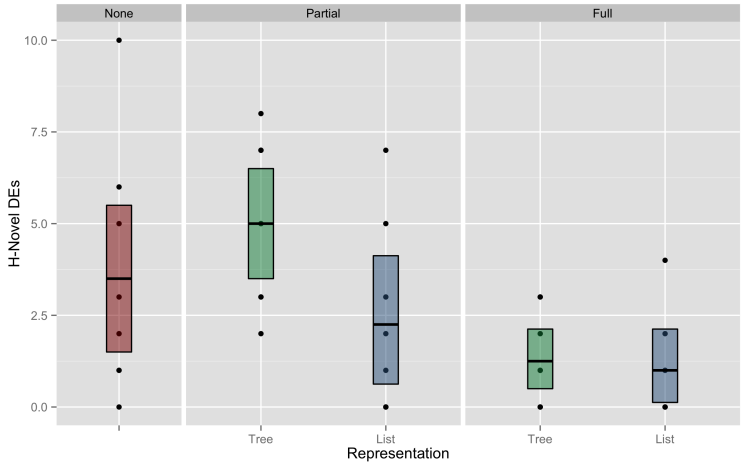


Figure 6-11: Novelty scores across conditions $(R = 1000, 95\% c.i.)$

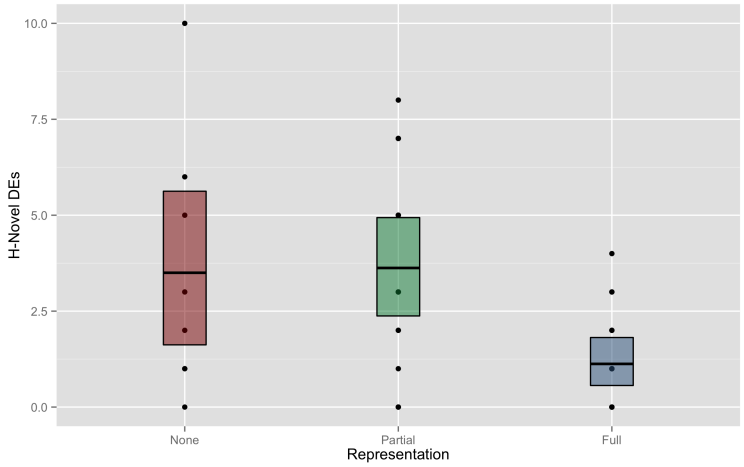


Figure 6-12: Novelty scores across levels of DS support $(R = 1000, 95\% c.i.)$

the lower the level of DS support provided, the greater the chance for historically novel design ideas for the domain. No difference was found in the number of historically novel design ideas for the domain, so we can accept the null hypothesis for DS representation and novelty, and accept $H_E(\mathbf{6})(\mathbf{b})$. The implications of these findings are discussed in Section 6.4.

6.3.3 Workload Analysis

As well as analysing the designs that were created as an output of the study by our participants, we sought to identify the workload that participants associated with the tasks, either that they perceived from the task itself, or that they chose to expend. The results of the NASA TLX subjective self-assessment were analysed following the same process as the analysis of the other aspects of participants' designs. That is, we first calculated the correlation between the test variable and the level of support provided by the DS, followed by individual analysis of the conditions to determine if there is significant difference in the test variable across DS representations.

As we will see, the overall TLX score reports some difference between conditions, and as such we will analyse each of the constituent measures of the TLX scores separately in order to identify what aspect of workload was reported to be higher in each case.

Mental Demand

Correlating DS support and weighted mental demand score indicated weak positive correlation for the tree representation, and no correlation for the list representation (tree: $rs[22] = 0.295$, $p = 0.162$; list: $rs[22] = 0.022$, $p = 0.918$). An ANOVA indicated no significant difference in perceived mental demand across conditions: $F(4, 35) = 1.742, p = 0.163$.; across representations: $F(2, 37) = 1.416, p = 0.256$; or levels of DS support: $F(2, 37) = 2.436, p = 0.101$.

Physical Demand

The Spearman's rank correlation coefficients for perceived physical demand score against level of DS support showed weak positive correlation in the tree representation and no correlation in the list representation (tree: $rs[22] = 0.369$, $p = 0.076$, list: $rs[22] = 0$, $p = 1$). There were a number of zero values that meant the data were unsuitable for a Shapiro-Wilks (S-W) test for normality, and hence could not be normal. Non-parametric bootstrapping showed no significant difference in physical demand scores. The results of this are shown in Figure G-3(a), which can be found in Appendix G.

Temporal Demand

We identified moderate positive correlation between DS support and perceived temporal demand in both DS representations (tree: $rs[22] = 0.479$, $p = 0.018$; list: $rs[22] = 0.38$, $p = 0.067$). The data were not normally distributed in condition 4 S-W($p = 0.013$). Non-parametric bootstrapping at 95% c.i. indicated no significant difference in temporal demand scores across conditions. The results of this are shown in Figure G-3(b), which can be found in Appendix G. The contradiction between these two analyses prompts further discussion of perceived temporal demand in our study.

Performance

Correlating perceived performance with the level of DS support identified weak negative correlation (tree: $rs[22] = -0.336$, $p = 0.109$; list: $rs[22] = -0.298$, $p = 0.156$). The data were not normally distributed in condition 4 S-W($p = 0.004$). Non-parametric bootstrapping at 95% c.i. indicated no significant difference in performance scores across conditions. The results of this are shown in Figure G-4, which can be found in Appendix G. The contradiction between the negative correlation yet no significant difference across conditions prompts further discussion of participants' perceived performance.

Effort

We identified strong positive correlation between perceived effort and DS support in the tree representation, and weak positive correlation in the list representation (tree: $rs[22] = 0.701$, $p < 0.001$; list: $rs[22] = 0.122$, $p = 0.571$). An ANOVA indicated significant difference in perceived effort across conditions: $F(4, 35) = 3.991, p = 0.009$, and Tukey's HSD indicated that there was significant difference in perceived effort score between condition 4 with each of conditions 1 ($p = 0.008$) and 5 ($p = 0.02$). Condition 4 had a higher perceived effort than conditions 1 and 5. An ANOVA indicated that there was no significant difference in effort across levels of DS support: $F(2, 37) = 2.216, p = 0.123$, whereas significant difference was identified across DS representations: $F(2, 37) = 4.746, p = 0.015$. Tukey's HSD identified significant difference between no explicit DS representation and tree-based representation.

Frustration

Spearman's rank correlation for frustration indicated no correlation between frustration and DS support (tree: $rs[22] = -0.044$, $p = 0.837$; list: $rs[22] = 0.033$, $p = 0.878$). An ANOVA indicated significant difference in frustration scores across conditions: $F(4, 35) = 2.926, p = 0.035$, and a Levene's test for homogeneity of variance indicated that variances

were heterogeneous: $(F = 7.23, p < 0.001)$. A Games-Howell post hoc test indicated significant difference between the tree representation at partial (condition 2) and full support (condition 4).

We calculated Z-scores for each of the data points and identified that condition 2 contains an outlier with $Z > 3$, and this outlier would have a strong effect on our conclusions when bootstrapping on the mean. Thus, we performed bootstrapping using the median frustration score, which is more resilient to outliers in the data. This identified no significant difference in frustration scores across conditions, and is shown in Figure G-5 in Appendix G.

TLX Score

Correlating the overall TLX score (a measure that incorporates all of the measures evaluated previously) against the level of DS support indicated moderate positive correlation for the tree-based representation, and no correlation for the list-based representation (tree: $rs[22] = 0.553$ $p = 0.005$; list: $rs[22] = 0.096$, $p = 0.656$). An ANOVA indicated significant significant difference in TLX score across conditions: $F(4, 35) = 3.729, p = 0.012$, and Tukey's HSD indicated that there was a significant difference between conditions 1 and 4 ($p = 0.022$). There was no significant difference in TLX scores across representations: $F(2, 37) = 2.354, p = 0.109$, or levels of DS support: $F(2, 37) = 1.912, p = 0.162$.

We have identified evidence that performance, effort, and temporal demand may have varied across the conditions of our study. The implications of these findings will be discussed in the next section.

6.4 Discussion

The aim of this study was to explore the effect that using design spaces can have on the the design ideas chosen by designers in the process of designing a software artefact. We asked participants to generate a conceptual design for an EUSC application with varying levels of support provided by a explicit design space in two different representations.

The designs that participants created in the study were all notionally valid, but not directly comparable with one another. There were a wide range of different designs created, some of which were more specific, and others were more general. The designs tended to either specifically focus on features and consider the domain of the design or task it should complete, whereas others focused more on the domain or task and less on the features the application would have. Given the difficulty of comparing these aspects of the designs, we compared them based on properties of designs in general.

Our study has numerous findings regarding the effect that varying the level of DS support and DS representation had on the designs that were produced by our participants: novice

designers with limited knowledge of the domain of the artefact to be designed. In this section, we discuss each of the properties that we used to evaluate the differences between the designs.

Our discussion of the findings of the study will be based around the impact produced by each of our independent variables: DS representation and DS support. First, our discussion focuses on the properties for which we did not identify any change across the conditions of the study.

We were able to demonstrate that all of the designs created by our participants were correct, consistent, and relevant. The most important of these is correctness, given that a design must be correct in order to ever be in a state that it can be implemented.

Consistency of the designs is also important since a design could not be implemented if design choices contradict one another. We did identify some inconsistent pairs of design choices, but did not have enough data to identify whether either DS support or DS representation had significantly more inconsistencies than the other. The presence of inconsistency at all means that we should suggest a mechanism for stopping designers making inconsistent choices using the tool. Since we were able to identify these inconsistencies in our participants' designs, this suggest that the design space creator could identify inconsistent choices whilst they are creating the explicit design space. These inconsistencies could then be indicated to the designer who is using the design tool for design generation.

All of the designs generated in the study were also relevant to the task that participants were asked to complete. This is likely to be due to the experiment being lab-based rather than a field study, since participants were given direct instructions to complete the task. Future work is required to identify if the relevance of the designs changes when the design tool is used for design generation in the real world rather than in a lab setting.

6.4.1 Impact of DS representation

We only rejected one null hypothesis related to the DS representation that was used to present the design space to participants: $H_E(\mathbf{5})(\mathbf{b})$, where we concluded that the representation of the DS affected the cleanliness of the design that was created by participants. Cleanliness refers to the balance of the design, which we measured across categories and more specific sub-categories that were derived independently of the work on the DS. We believe cleanliness is important because it is important that a conceptual design considers the important topics of the domain in which the design is created, which in the case of EUSC are reflected in the categories and sub-categories that we measured the designs against.

Our findings indicated that designs had a better balance score when participants were presented with a tree-based representation of the DS rather than a list-based representation. On one hand, this is perhaps not surprising since a tree-based representation conveys some

idea of balance implicitly – participants might choose some elements from one branch and then move on to the next branch in a systematic way. However, as we will see below, neither of the analyses of the balance of the design that we performed explicitly account for this behaviour.

The first of our analyses of balance was measured across categories of the explicit design space. Participants were provided with separate trees or lists for the different categories of the design space, so the representation of that category should not have had a different effect on whether participants were likely to create a design with a higher or lower balance score. It is possible that the structure present in the tree-based representation of the explicit design space motivated our participants to use a more structured thought process than those who were not supported by the design space.

The second analysis was based on sub-categories that were identified independently to the creation of the explicit design space. Since the categories did not match the contents of the design space in either representation, there is no obvious reason why either the tree- or list-based representation might affect the balance score for sub-categories. It is possible that the explicit structure that is represented in the tree-based representation meant that participants were more likely to select design choices across a breadth of topics in the domain.

Since we only rejected one null hypothesis related to the DS representation, we only have a single piece of evidence as to which of the two representations should be used in the tool in future. This limited evidence shows that the tree-based representation might help users to generate a more balanced set of design ideas. We believe that this is reasonable since the tree-based representation provides more information than the list-based representation.

6.4.2 Impact of DS support

We rejected three null hypotheses related to the level of DS support that participants were provided, and accepted the following experimental hypotheses:

- $H_E(\mathbf{4})$: The level of DS support does not affect the generality of the designs created by participants.
- $H_E(\mathbf{6})$: The level of DS support does not affect the cleanliness of the designs created by participants.
- $H_E(\mathbf{7})$: The level of DS support does not affect the number of historically novel design choices in the design.

We will discuss each of these separately, before providing an overall discussion of the effect of varying the level of DS support that was provided to participants.

Generality

Our evaluation of the generality of the designs was broken down into three sub-measurements: abstractness, specificity and complexity. We identified correlations and significant differences between each of these measurements across the level of DS support. A summary of our findings is shown in Table 6.4.

Table 6.4: The effects of varying DS support across the measures of design generality.

DS Support	Less	More
Abstractness	More abstract	More concrete
Specificity	Slightly less specific	Slightly more specific
Complexity (# elements)	Fewer	More
Complexity (# causal relations)	Fewer	More

Our participants' designs were more concrete when more DS support was provided, and more abstract when they were provided with less support. We cannot say exactly how abstract or concrete the designs are, given that abstractness is a relative relationship. Clearly before it can be implemented, a design needs to be concrete, but in the very early stages of design, it is not clear whether this concreteness is required.

The designs were also more specific with higher levels of DS support, which is unsurprising as specificity is clearly related to abstractness: an artefact cannot be abstract whilst at the same time being specific. In order to implement a design, it cannot be vague, although similarly to abstractness, it is not clear how specific a design needs to be in the conceptual phase.

Complexity was also shown to be higher when participants were provided with a higher level of DS support. We identified an increased complexity with increased DS support both in terms of the number of design choices in designs and the number of causal relationships between the design choices in designs. It stands to reason that the more design choices in the design, the more likely there are to be more causal relationships between these design choices. However, it is not clear whether a design being more complex is good or bad in this case.

Thus, the level of DS support can be varied by the designer to help to achieve the desired level of generality in the design. However, abstractness, specificity, and complexity cannot be controlled independently. The implications of this for the design space tool will be discussed in Section 6.5.

Cleanliness

We identified that designs were more balanced when more DS support was provided, both when the size of the design was taken into account, and when it was not. For the size-

dependent measurement, we believe that we have evidence that the balance score was overpowered by the number of design choices in the design, and that this may simply be reporting the findings of the detail complexity analysis (based on the number of design choices in the design) rather than taking the balance into account. Thus, we should be tentative about what conclusions we can draw from this result.

When we considered the size-agnostic calculation that did not take the number of design choices into account, we also found an increase in balance scores when DS support was increased. This finding is interesting, because it shows that in higher levels of DS support, our participants considered the breadth of topics in EUSC and software engineering in a more balanced way than those who were provided with lower levels of DS support.

It has been claimed that novice behaviour in problem solving is normally reflected by a 'depth-first' approach, whereas experts would normally use a top-down or breadth-first process [Cross, 2004]. We suggest that the higher balance score indicates breadth-first thinking, since participants with higher balance scores have given more equal consideration to a breadth of topics within EUSC and software engineering. Our participants were all novices, yet we have some evidence that our design tool and design space helped them to operate with behaviour that is not consistent with novice designers.

Furthermore, very few of our participants had any experience with EUSC applications, and none reported that they used them frequently. Hence the provision of DS support helps novice designers to create designs that are more developed than would be expected, especially considering their lack of knowledge of EUSC – the domain in which the design was being created.

Novelty

We identified a decrease in the number of historically novel design ideas that participants identified as the level of DS support was increased. This is the first of our results that clearly indicates a better result with lower levels of DS support.

We believe that the presence of the solutions in the design space may have constrained participants thought process in generating new designs. It would seem sensible to assume that providing participants with a large number of possible design solutions, participants would be less likely to identify their own solutions at all. It may then be likely that the more custom choices made, the more likely they are to be some historically novel ideas in that collection of design choices.

This is the the first finding of the study that provides some evidence that a *lack* of DS support may be beneficial in the creation of design artefacts since a lack of support may encourage designers to make design choices that are novel in the domain. Thus, any future work would need to take this into account. We will discuss our recommendations for future use of a

design space tool in Section 6.5.

The identification of novel design choices in this study meant that we were able to extend the design space to contain these new, novel design elements that were generated by our participants.

6.4.3 Workload

The results of our NASA TLX indicated that there was some evidence for a difference in perceived workload across conditions, in particular between condition 1 (no support) and condition 4 (full support, tree-based representation). Since these conditions differed both in representation and level of DS support, it isn't clear which variable affected the workload score, either perceived by the participant, or chosen by them. To identify which component of the TLX score might have had this effect, we broke the workload score down into its constituent components, and evaluated each of these separately.

We identified that further discussion was needed for the effort component of the TLX with respect to the levels of DS support, and in particular that participants using the tree-based representation and full support reported a higher effort score than participants who had no DS support. We believe that rather than the task requiring more effort, participants *chose* to expend more effort because there was a much clearer method for them to complete the task than conditions with no support, and as such they would have been more likely to expend effort to complete it.

We also identified that perceived performance declined as level of DS support increased, where temporal demand increased. We believe that perceived temporal demand increased with DS support because the task being undertaken was much clearer in higher levels of support, meaning that participants were more aware of the scope of the task. Those in lower support or no support, however, would have had less awareness of the scope of the design that they might want to create. We believe that a decline in perceived performance was also due to participants being aware of alternatives to the decisions that they made, and hence might worry that they had not made the right decisions.

Whilst we have seen some deviation in the TLX and its constituent measures across the different conditions of our study, the evidence for where exactly this occurs is relatively weak. Where difference is evident, it often relates to conditions 4 (full support, tree-based representation), and 1 (no support). We suggest that this is because condition 4 was effective at conveying the size and structure of the whole concrete design space, which may have been overwhelming for our participants. The next section presents our recommendations for dealing with problems such as this in future uses of the design tool.

The findings of our study suggest that design spaces can help to focus the ideas that designers generate, and identify topics that the designer might want to consider. However, aiding

designers with this structure also has the knock on effect of restricting these ideas to the concepts presented in the design space.

6.5 Design Implications for Design Space Tools

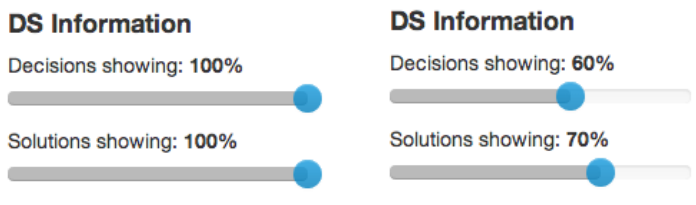
We have some evidence that our participants created designs that were more balanced using the tree-based DS representation, and found no difference between any of the other properties or measures in either representation. Objectively, the tree-based DS representation is better able to present both the design elements in the explicit design space and the relations between those design elements, whereas the list is much not able to convey the relations between design elements as effectively. Thus, we recommend the tree-based design space representation as the better on for using the design tool to generate designs.

We identified several findings relating to the level of DS support participants were provided with and the design ideas that they generated. A summary of our findings is presented in Table 6.5.

Table 6.5: General implications of varying DS support.

DS Support	Less	More
Abstractness	More abstract	More concrete
Specificity	Slightly less specific	Slightly more specific
Complexity (# elements)	Fewer	More
Complexity (# causal relations)	Fewer	More
Balance	Less balanced	More balanced
Novel ideas	More	Fewer

None of the design properties in Table 6.5 are clear cut as to whether they are 'good' or 'bad', except perhaps novelty. We argue that the design space tool can be modified to take note of the different aspects of these findings, and use them to designers' advantage. To account for these general implications, we suggest that the design space tool be re-designed to allow the designer to be able to control the level of DS support with which they are provided.



(a) Whole explicit design space (b) Reduced information

Figure 6-13: DS support slider in the design space tool.

Figure 6-13 shows the sliders to control DS support that we implemented in the final version of the design space tool described in Chapter 5. The sliders allow the designer to incrementally reduce the amount of information contained within the design space, in terms of the decisions and solutions that are presented in the explicit design space. To relate these sliders to the levels of DS support used in the study, full support reflected 100% on both sliders, partial support would have 100% on the decision slider and 0% on the solution slider, and no support would have 0% on both sliders.

Using this control over the DS support that is provided, designers are able to take advantage of the findings in the study. For example, a designer would be able to start off the process with no support in order to generate a number of abstract, novel design choices, before increasing the support provided to them, and adapt the design to be more specific, better balanced, and more concrete.

We believe that varying the DS support in this way also mitigates the problems identified in participants' perceived workload. If a designer feels overwhelmed, they can simply reduce the amount of information that is being presented to them until they are comfortable. They can then raise the amount of information presented to them incrementally.

This should ensure that the designs created using the tool benefit from the different findings of our study and be able to actively improve the design they create as a result. The learning effect that we sought to avoid to ensure our findings is not of consequence in the real world, and may even be helpful to the design task when using the design space tool.

6.6 Limitations and Future Work

The work presented in this chapter was a laboratory study and this decision meant that we sacrificed the knowledge of how the design tool and design spaces could be used by designers in real-world situations. This limitation was also manifested in our focus on novice designers. We decided to focus on novice designers because we felt that they would receive the greatest benefit from the support that is provided by the design space, and our finding relating to our participants' behaviour reinforces this decision.

This prompts an avenue for future work: performing the same study with professional designers rather than novices. To be directly comparable with the results of this study, this would need to be carried out in a lab setting, and use the same task that our participants were asked to complete. The goal of this study would be to identify whether professional designers receive the same benefits and drawbacks from design space support as novice designers.

Future work could also include evaluating how the design tool can be used in the real world. We would assume that the designer is a professional, or that they have their own design task that they are seeking to complete, meaning that it would be much more difficult to conclude

that any observed effects are caused by the design tool and not the other variations that would be inherent from a field study-based experiment.

Evaluating how the design tool could be used in a different domain would help to validate the findings of this study, and ensure that the findings are not specific to EUSC. We have already highlighted that design spaces are most suitable for use in design problems within ill-structured domains: those where there are many solutions to the problem and many ways of reaching these different solutions.

We were only able to recruit 8 participants per condition of the study, which limits the statistical power of the conclusions that we made. We argue that the amount of data gathered across all of the participants in a single condition mitigates this somewhat. We were restricted in the scope of the participants that we could gather because of the requirement that they have had some experience designing software.

6.7 Chapter Summary

This chapter sought to address our second research goal: **RG2 Create and evaluate a design space for EUSC applications.** We documented the creation of the explicit design space in Chapter 4, which incorporated the historically novel design choices that participants in this study produced. This chapter focused on the evaluation of how the level of support provided by a design space can affect the designs that are created by participants.

We presented a study that we carried out in order to assess how design spaces can be used in the process of design generation. Our participants created a conceptual design (a list of important features or concepts) for an EUSC application, which they recorded using an early prototype of the design tool described in Chapter 5.

We varied the level of support that users were provided with across three levels: no support (no design space), partial support (only design decisions), and full support (design decisions and potential solutions). We also varied the representation of the design space across two levels: a tree-based representation and a list-based representation.

We identified that comparing the designs on face value would be very difficult due to the breadth of topics that the designs contained, as well as the different focus that participants had for their designs. Instead, we evaluated designs based on a collection of general properties that designs have [Sinderen, 1995], and the novelty of the ideas in design. The properties against which the designs were evaluated were: correctness, consistency, propriety, generality, and cleanliness.

We evaluated each of the designs based on these properties, and found that varying the DS representation did not have any significant effect on these properties. We identified links between the level of DS support and the generality, cleanliness, and novelty of the

designs. Designs generated under higher levels of support were shown to be less general, more balanced, and less novel.

Thus, we suggested that design spaces help to structure the thought process of the designer, reflecting the increase in balance, specificity, correctness, and complexity. However, this structure inhibits designers' ability to generate novel design ideas.

These findings have several implications for the future use of design spaces in the design process. Arguably the most important of these findings was that an increase in support from the design space yielded fewer historically novel design ideas, meaning that design spaces should not be the only resource used by the designer. Our recommendation to resolve this was to ensure that designers start with no support from the design space to ensure that they are given the opportunity to generate novel ideas before being presented with the design space.

The cleanliness of the design also increased with the level of DS support that was provided, meaning that participants considered each of the categories of the design space equally, as well as giving equal consideration to topics within the domains of EUSC and software engineering.

The generality of the design was evaluated by measuring the relative abstractness and specificity of each of the designs, as well as their complexity. We identified that designs created when participants had more DS support were more concrete, more specific and more complex. Whilst it is not entirely clear the effect that these properties might have on designs, it is clear that before being implemented a design would need to be concrete and specific. Thus, a design being more abstract and less specific might be more useful in the early part of design, specificity and concreteness are useful in the later stages.

Our evaluation of novice designers using the design tool to generate designs for EUSC applications identified that varying the level of design space support provided has a number of effects on the designs created by participants, and by controlling the design space support, we can take advantage of the positive aspect of these changes.

Furthermore, we were able to demonstrate the use of (an early prototype of) the design generation module of our design space tool. And whilst we are not able to come to a conclusion as to how well it supports the design generation task, we know that 40 novice designers were able to use it to generate designs that were all correct and relevant.

CHAPTER 7
CONCLUSION

7.1 Thesis Summary

This thesis aimed to explore design spaces and how they can be used to inform the design of End-user Service Composition (EUSC) applications. EUSC is an instance of an ill-specified problem: a problem with multiple solutions, multiple ways of reaching these solutions, and no definitive method for identifying which solution is best [Jonassen, 1997].

To achieve this aim, we identified the following research goals:

RG1: Derive and evaluate a set of requirements for an EUSC application.
RG2: Create and evaluate a design space for EUSC applications.
RG3: Create and evaluate a design space Tool to facilitate the design and creation of design spaces and domain applications.

RG1 was addressed in Chapter 3. We presented a study in which we derived a large set of requirements for an EUSC application from potential end-users.

We identified an established requirements gathering method as the best base upon which to build our custom method: the Scenario-based Requirements Analysis Method (SCRAM). We made a number of modifications to SCRAM to make it suitable to use end-users as participants, one of which was to move the requirements elicitation step outside of the participant-facing sessions.

Directed Content Analysis (DCA) was selected to analyse the output of the study sessions, as it was resilient to the priming of the topics of the questions posed to participants in the interview. This content analysis yielded a large list of topics which we then used to elicit a set of requirements.

We evaluated the requirements that we gathered based on their completeness, consistency, and correctness. We verified that the requirements were individually complete, the functional requirements were complete, and that the non-functional requirements were identified as being incomplete when evaluated against ISO/IEC 25010 [ISO/IEC, 2008]. Our requirements

validated existing requirements in the domain, as well as around half being evident in currently available EUSC applications. The vast majority of the requirements could be used in their current state as part of the specification of an EUSC application, whereas some require more research before they are eligible for use.

RG2 was addressed in Chapters 4 and 6. In Chapter 4 we documented the creation of an explicit design space for EUSC applications using a novel, bespoke method. We extend existing definitions of design space to provide a more well-defined vocabulary for describing what design spaces are and how they can be used. Additionally we present an evaluation of the method by which the design space was created.

In the literature review, we identified that the vocabulary used to describe design spaces is not clear, particularly when we consider the difference between the metaphorical 'design space' for a software artefact, and a concrete model of design choices like those presented in a number of design spaces for EUSC applications [Aghaee et al., 2012, Mehandjiev and De Angeli, 2012], etc.

We provided two definitions for types of design space to address this deficiency in terminology based on the scope or abstractness of the design space:

- **Implicit Design Space**: The theoretically infinite space containing *all* possible design decisions for a particular design problem. Since it is theoretically infinite in size, the implicit space cannot be fully described [Westerlund, 2005].
- **Explicit Design Space**: A finite model of the implicit design space, restricting decisions based on some criteria. Aspects of design solutions can be modelled in a explicit design space. Since we cannot describe the abstract space, creating a concrete model is a necessary step in describing design spaces and making them usable.

We also extend our discussion to how explicit design spaces can be used, and where they are used in the software engineering process. These uses include evaluation of current designs, and the generation of new designs.

Existing work on design spaces gives little guidance as to how design spaces should be created, particularly those in the EUSC domain. [MacLean et al., 1991] provide a list of heuristics that can help with the process, but do not provide a method to specify how these heuristics should be applied.

We created a method that takes advantage of the resources available in the EUSC domain. These resources were split into two sets: the first set were based on artefacts that could be reviewed by a single researcher, and the second set required some form of research or study using participants where artefacts were not available to be reviewed. The generalisability of our design space creation method depends on the resources that are available in the domain – not all domains will have the wealth of other resources that we did with EUSC. When resources are not available, the participant-facing stages become more important.

We demonstrated our design space creation method by using it to create an explicit design space for EUSC applications. Carrying out our method yielded a design space containing over 600 design elements (see Appendix D), which is split into four categories:

- **Functional:** Design choices relating to the functions that an EUSC application can perform (excluding specific interactions with entities).
- **Non-functional:** Design choices relating to the non-functional aspects of an EUSC application such as representation, target user, etc.
- **Structural:** Design choices relating to the structure or architecture of the application. In the EUSC case this also contains design choices relating to the composition architecture/structure.
- **Entity:** Design choices directly relating to entities that the user can interact with in the application. In the case of SC this contained interactions with and representations of services – both components and composites.

In Chapter 6, we evaluated how the design space can be used in the process of design generation by asking novice designers to generate a conceptual design for an EUSC application using an early prototype of the design space tool. We varied the level of design space support across three levels – no support, partial support and full support – and the representation of the design space – tree-based representation and list-based representation – between participants.

Since we could not evaluate participants' designs directly against one another, we evaluated them with respect to a number of general properties of designs. A summary of the findings of the study are presented in Table 7.1.

Table 7.1: Design changes with DS support

DS Support	Low	High
Abstractness	More abstract	More concrete
Specificity	Slightly less specific	Slightly more specific
Complexity (# elements)	Fewer	More
Complexity (# causal relations)	Fewer	More
Balance	Less balanced	More balanced
Novel ideas	More	Fewer

RG3 was addressed in Chapter 5, in which we presented the creation of a design tool that supports three activities related to design spaces: design space creation, application profiling and design generation. The tool was evaluated analytically to identify how well the design tasks are supported.

[Baum et al., 2000] suggests tool support as being particularly important when using design spaces because of their potential vast size. We created our design space tool to support three tasks that relate to the use of design spaces:

1. **Design space creation:** The design space tool can be used to record the creation of an explicit design space by allowing the user to add and link collections of design decisions and potential solutions.
2. **Application profiling:** The design space tool can be used to track the design choices that are made in applications in the domain by allowing the user to add new applications that they are profiling, and select the elements of the design space that are evident in the design of those applications.
3. **Design generation:** The design space tool can be used to create a new design for an application in the domain. The designer is able to select design choices from the explicit design space, or choose their own custom design choice. The tool also allows the designer to record the rationale for each decision for traceability later in the design process.

We analytically evaluated how the design tool supported each of these tasks using hierarchical task analysis and a cognitive walkthrough. The cognitive walkthrough found a minimal number of usability problems that were subsequently fixed. We were able to use hierarchical task analysis to identify that the design space tool effectively supported the tasks of design space creation and application profiling. We were unable to identify how effectively the design space tool supported the design generation task due to the lack of specificity in the task of generating new designs using an explicit design space. However, in Chapter 6 we were able to demonstrate that the design space tool supported the design generation task with a group of novice designers.

7.2 Findings and Contributions

In this section we present the findings and contributions of the work in this thesis.

7.2.1 RG1: Derive and evaluate a set of requirements for an EUSC application.

Contribution 1: An end-user focused end-to-end requirements gathering method. Established requirements gathering methods focus on gathering requirements from business customers rather than end-users or consumers. We developed a new method for gathering requirements from a set of end-users in a given domain, which combines an established requirements gathering method: SCRAM, directed content analysis, and a bespoke requirements elicitation step. We demonstrated the use of this method in the domain of EUSC and gathered 139 requirements, where previous approaches had gathered very few.

We demonstrated that all of the requirements were correct and consistent, and that the functional requirements were functionally complete. Whilst the non-functional requirements

were not identified as being complete, we suggested adaptations to the method to mitigate this.

This method provides requirements engineers a way to derive requirements from a consumer facing domain, where there is no single business customer who can provide a concrete set of answers to how the application should operate. Furthermore, the method is robust to domains with which the end-user is unfamiliar before taking part in sessions, as is demonstrated by a number of the participants of our requirements gathering study.

Contribution 2: 139 requirements for an EUSC application. We identified two prior requirements gathering approaches in EUSC, as well as a number of small sets of requirements reported for EUSC and mashup applications in prior literature. Some of this small number of requirements could be used in the specification of an EUSC application, whereas others highlight areas in the domain in which future work is required. We sought to identify a large set of requirements that could be used to specify an EUSC application.

We derived a set of 139 requirements for an EUSC application, which we demonstrated were correct, consistent and functionally complete. The whole set of requirements are listed in Appendix C.

The set of requirements we gathered is by far the largest set of requirements for an EUSC application identified to date. A number of our requirements validate others that have been identified by other authors, as well as roughly half being evident in currently available EUSC applications. Furthermore, we demonstrated their validity in the domain by incorporating them into our explicit design space for EUSC applications. In their current state, the requirements could be used in the specification of an EUSC application, as well as a number which identify interesting areas of future work in EUSC, mashups and SC.

7.2.2 RG2: Create and evaluate a Design Space for EUSC applications.

Contribution 3: A method to create explicit design spaces. Existing design spaces, particularly those in EUSC, provide very little insight as to how they were created. Where methods or heuristics are provided, they are either very vague or do not give guidance as to exactly how these methods and heuristics should be applied.

We devised, demonstrated and evaluated a method for creating explicit design spaces in a given domain. The method has 5 distinct stages:

1. Collation of existing explicit design spaces
2. General literature review of SC, EUSC, and sub-domains thereof
3. Application review of EUSC applications
4. Requirements gathering study
5. Design space evaluation study

The specification of a method for creating a design space in a given domain makes design spaces a more attractive proposition for other designers and researchers, as there is a defined method that they can follow to create a concrete design space. We discussed the generalisability of the method to other domains, and identify how each stage could be used in a different domain, where different resources are available. We also demonstrated that the method could be used to create a design space in the domain of EUSC.

Contribution 4: An explicit design space for EUSC applications. One of the research goals of this work was to create and evaluate an explicit design space for EUSC applications. We identified this as a research goal based on prior design spaces from the EUSC domain, and the nature of EUSC as an ill-structured problem.

We created an explicit design space for EUSC applications that contained over 600 design elements, grouped into four categories: functional, non-functional, structural and entities.

The design space we created is much larger than any previous explicit design space for EUSC applications. Furthermore, we demonstrated how this design space could be used in the design process for an EUSC application, as well as integrating the results of this evaluation to subsequently improve the design space further.

Contribution 5: An understanding of the effect of design spaces on the design process for novice designers. The evaluation of the explicit design space is an important part of our research goal related to the explicit design space. In particular, we sought to explore to what extent our design space could support the process of generating a design for an EUSC application, and what specific effects the design space might have.

We identified that increasing the level of design space support in the design process has the following effects:

- Fewer historically novel design ideas
- A more balanced design
- More concrete
- More specific
- Larger
- More connections between design elements

The results of this study contribute insight into how design spaces and the tools that support them can be used to generate designs in a domain. We contribute an evaluated design space in the domain of EUSC. Furthermore, we argue that the results identified here would be generalisable to other domains thus contribute a general design tool for creating design spaces, profiling applications, and generating designs.

7.2.3 RG3: Create and evaluate a Design Space Tool to facilitate the design and creation of design spaces and domain applications.

Contribution 6: A design tool for creating design spaces, profiling applications, and generating designs. Given their potentially vast size, explicit design spaces require support from tools for designers to be able to use them effectively in any part of the design process [Baum et al., 2000]. Such tools would need to support each of the different interactions that we have identified as being important when using design paces for design.

We created a design tool that supports designers in creating design spaces, profiling applications in the domain, and generating new designs. We evaluated the design tool analytically to identify how well it supported the tasks for which it was designed to support. We were able to use hierarchical task analysis to identify that the design space tool provides adequate support to the tasks of design space creation and application profiling. We also demonstrated that the design space tool can be used for design generation in our exploration of how design spaces can be used in the design process.

7.3 Limitations

Both of the studies that we carried out suffered from limitations regarding the number of participants across which the study was run. The requirements gathering study was performed with 10 participants (5 male, 5 female), and with a number of different jobs and different levels of experience. 10 participants was deemed suitable because of the volume of data that were gathered in the earlier sessions, and the length of these sessions (1.5-2 hours). The second study had considerably more participants overall (40), but these participants were spread across 5 conditions (8 per condition). Our requirements for the participants in the design study were much higher than the requirements gathering study, since we needed participants to have had limited prior experience in designing software.

Both of the studies that we carried out were lab-based, which may limit their generalisability to real-world situations. Since our requirements gathering study was focused on end-users, we do not believe that the results would have been different if it were carried out outside a lab. We also suggest that the findings related to the designs generated in the design space study would not have been different if the study were carried out in a real-world situation.

The design space creation method that we present contains a number of stages that are only applicable if certain resources are available in the domain in which the explicit design space is being created. This dependence on resources impacts the generalisability of the method as a whole, but due to the breakdown of the method into these stages, we are able to suggest how the method could be applied in domains where these resources may not be available to the designer or researcher.

The majority of the mashup tools that were reviewed in the prior EUSC design spaces have since closed down, which might highlight a lack of interest in the domain. However, as we have discussed, more general EUSC applications have been shown to be popular, particularly on Android. Service Composition, and hence EUSC also draws parallels to a relatively new research area: the Internet of Things (e.g. [Liu et al., 2012]).

7.4 Future Work

Avenues of future work exist in different aspects of each of the three major bodies of work within this thesis: the requirements and the method through which they were derived, the explicit design space for EUSC applications as well as its creation and evaluation, and the design tool.

The requirements that we gathered in Chapter 3 could be used a specification for an EUSC application, which could subsequently be designed and implemented. We did not use the requirements to specify an EUSC application ourselves because our research goals directed us towards work on design spaces. However, the vast majority of the requirements are in a state where they could be used directly as part of such a specification.

We believe that some of the requirements we gathered are not directly usable in a specification in their current state, as more research would be required before they could be used as part of such a specification. An example of an interesting requirement requiring future work is that the EUSC application should automatically identify potential compositions by profiling user behaviour. This would require investigation to identify whether it is technically possible, since it is not clear how the application would be able to 'watch' what the user does, and furthermore how it would determine what tasks could be performed with a composite services instead of a manual process.

The method that we used to generate the requirements could be validated by performing it in a domain other than EUSC. We provide a discussion of the generalisability of our requirements gathering method in Chapter 3, but to ensure that it is generalisable it would need to be tested in another end-user focussed domain.

We used a bespoke method to create our explicit design space for EUSC applications, and argued that this method is generalisable to other domains. To validate this assertion, we suggest that this method should be applied in domains with varying levels of resources available so that future researchers can identify how varying resources affect the application of the different stages of our method.

As with our set of requirements, there are a number of concepts within our explicit design space for EUSC applications that prompt future work, including the one considered above in the future work for requirements. Any avenues for future work within the EUSC design

space were derived from either the requirements gathering study or the design space study, since those found in other stages had either already been implemented or researched.

Our study into the exploration of the effects of using a design space in the design generation process was carried out using our explicit design space for EUSC, and we suggest that it could be carried out using a different domain to verify our results. That is, we found an increase in specificity and concreteness with increased design space support, as well as a decrease in novelty. These findings could be verified both through the use of a design space in a different domain, and through performing this study again with professional designers.

A number of the designs that participants generated in our design space study are in a state where they are immediately usable as an initial specification for the design of an EUSC application, and as such could be fleshed out and implemented with relatively little extra work.

The final set of future work that is evident from this thesis is further evaluation of our design space tool. We were able to evaluate and demonstrate the use of the different modules of the design space tool in the EUSC domain and with novice designers. Future work could evaluate the design tool both in different domains, and with professional designers using the tool instead of novices.

7.5 Closing Remarks

The work in this thesis is based in two areas: End-user Service Composition (EUSC) and design spaces, and in particular how design spaces can be used to help to design applications that support EUSC.

We have extended the current state of knowledge in EUSC by deriving a set of 140 requirements for an EUSC application (available in Appendix C), and the creation of a explicit design space for EUSC applications that contains over 600 design elements (available in Appendix D). These resources can be used by other researchers, designers and software engineers to evaluate existing EUSC applications, identify future work in the domain, and ultimately to create new EUSC applications across a myriad of new and interesting domains.

On the way to making these artefacts, we developed a new method for deriving requirements from end-users based on the use of scenarios and a prototypical demonstrator. Furthermore, we constructed a method for the creation of a explicit design space where none existed previously, and demonstrated this method in our chosen domain of EUSC. To facilitate the use of these design spaces, we implemented a design space tool that allows designers to create design spaces, profile applications and generate new designs in the chosen domain.

We believe that both design spaces and EUSC have a bright future, both in research and in industry. Design spaces have been under-defined and underused in prior work, and we

believe our work helps provide a scaffolding for their use in future. Our work complements the rise in popularity of EUSC applications and related domains such as the Internet of Things, and as such can motivate future research in the domain.

BIBLIOGRAPHY

[Aghaee et al., 2012] Aghaee, S., Nowak, M., and Pautasso, C. (2012). Reusable decision space for mashup tool design. In *Proc. 4th ACM SIGCHI symp. on Engineering Interactive Computing Systems - EICS '12*, pages 211–220, New York, New York, USA. ACM Press.

[Aghaee and Pautasso, 2012] Aghaee, S. and Pautasso, C. (2012). End-User Programming for Web Mashups Open Research Challenges. *Current Trends in Web Engineering*, pages 347–351.

[Aghaee and Pautasso, 2013] Aghaee, S. and Pautasso, C. (2013). Live mashup tools: Challenges and opportunities. *2013 1st International Workshop on Live Programming (LIVE)*, pages 1–4.

[Aghaee and Pautasso, 2014] Aghaee, S. and Pautasso, C. (2014). End-User Development of Mashups with NaturalMash. *Journal of Visual Languages & Computing*, pages 1–19.

[Akehurst and Gadrey, 1988] Akehurst, G. and Gadrey, J. (1988). *The Economics of services*. Routledge.

[Al-Sharawneh and Williams, 2009] Al-Sharawneh, J. and Williams, M. A. (2009). A social network approach in Semantic Web Services Selection using Follow the Leader behavior. In *Enterprise Distributed Object Computing Conference Workshops, 2009. EDOCW 2009. 13th*, pages 310–319.

[Albinola et al., 2009] Albinola, M., Baresi, L., Carcano, M., and Guinea, S. (2009). Mashlight: a Lightweight Mashup Framework for Everyone. In *Workshop on Mahups, Enterprise Mashups and Lightweight Composition on the Web*.

[Albreshne and Pasquier, 2011] Albreshne, A. and Pasquier, J. (2011). A Template-Based Semi-Automatic Web Services Composition Framework. *Unpublished*.

[Alexander, 1964] Alexander, C. (1964). *Notes on the Synthesis of Form*. Harvard University Press, Cambridge, MA.

[Amabile, 1983] Amabile, T. (1983). The social psychology of creativity: A componential conceptualization. *Journal of personality and social psychology*, 45(2):357–376.

[Atwood et al., 2002] Atwood, M. E., McCain, K. W., and Williams, J. C. (2002). How does the design community think about design? *Proceedings of the conference on Designing interactive systems processes, practices, methods, and techniques - DIS '02*, page 125.

[Baum et al., 2000] Baum, L., Becker, M., Geyer, L., and Molter, G. (2000). Mapping Requirements to Reusable Components using Design Spaces. *Proc. 4th Intl Conference on Requirements Engineering*, pages 159–167.

[Baum et al., 1998] Baum, L., Geyer, L., and Molter, G. (1998). Architecture-centric software development based on extended design spaces. *Development and Evolution of Software Architectures for Product Families*, pages 197–204.

[Beckman, 2007] Beckman, S. L. (2007). Innovation as a Learning Process: Embedding Design Thinking. *California Management Review*, 50(1):25–56.

[Blackwell, 2002] Blackwell, A. (2002). First steps in programming: A rationale for attention investment models. *Proc. IEEE Symposia on Human Centric Computing Languages and Environments*, pages 2–10.

[Boden, 1996] Boden, M. A. (1996). *Dimensions of Creativity*. The MIT Press.

[Bottaro et al., 2007] Bottaro, A., Gerodolle, A., and Lalanda, P. (2007). Pervasive Service Composition in the Home Network. In *Advanced Information Networking and Applications, 2007. AINA '07. 21st International Conference on*, pages 596–603.

[Brønsted et al., 2010] Brønsted, J., Hansen, K. M., and Ingstrup, M. (2010). Service Composition Issues in Pervasive Computing. *Pervasive Computing, IEEE*, 9(1):62–70.

[Brunetti and Golob, 2000] Brunetti, G. and Golob, B. (2000). A feature-based approach towards an integrated product model including conceptual design information. *Computer-Aided Design*, 32(14):877–887.

[Cappiello et al., 2011a] Cappiello, C., Daniel, F., Matera, M., Picozzi, M., and Weiss, M. (2011a). Enabling End User Development through Mashups: Requirements, Abstractions and Innovation Toolkits. In Costabile, M., Dittrich, Y., Fischer, G., and Piccinno, A., editors, *End-User Development*, volume 6654, pages 9–24. Springer Berlin / Heidelberg.

[Cappiello et al., 2011b] Cappiello, C., Matera, M., Picozzi, M., and Sprega, G. (2011b). DashMash : A Mashup Environment for End User Development. *Web Engineering*, pages 152–166.

[Carroll et al., 1979] Carroll, J. M., Thomas, J. C., and Malhotra, A. (1979). Clinical-experimental analysis of design problem solving. *Design Studies*, 1(2):84–92.

[Casati, 2011] Casati, F. (2011). How End-User Development Will Save Composition Technologies from Their Continuing Failures. *End-User Development*, pages 4–6.

[Christiaans and Venselaar, 2005] Christiaans, H. and Venselaar, K. (2005). Creativity in design engineering and the role of knowledge: Modelling the expert. *International Journal of Technology and Design Education*, 15(3):217–236.

[Coyne, 1995] Coyne, R. (1995). *Designing Information Technology in the Postmoden Age*. MIT Press, Cambridge, MA.

[Cross, 2004] Cross, N. (2004). Expertise in design: an overview. *Design Studies*, 25(5):427–441.

[Csikszentmihalyi, 1997] Csikszentmihalyi, M. (1997). *Flow and the Psychology of Discovery and Invention*. Harper Perennial, New York, New York, USA.

[Curtis et al., 2001] Curtis, J. R., Wenrich, M. D., Carline, J. D., Shannon, S. E., Ambrozy, D. M., and Ramsey, P. G. (2001). Understanding physicians' skills at providing end-of-life care. *Journal of General Internal Medicine*, 16(1):41–49.

[Czarnecki and Eisenecker, 1999] Czarnecki, K. and Eisenecker, U. (1999). Components and generative programming. In Nierstrasz, O. and Lemoine, M., editors, *Proceedings of the Joint European Software Engineering Conference and ACM SIGSOFT International Symposium on the Foundations of Software Engineering (ESEC/FSE'99, Toulouse, France, September 1999)*, pages 2–19. Springer-Verlag.

[da Silva et al., 2009] da Silva, E., Ferreira Pires, L., and van Sinderen, M. (2009). Supporting Dynamic Service Composition at Runtime based on End-user Requirements. In *CEUR Workshop Proc.*

[da Silva et al., 2008] da Silva, E., Lopez, J. M., Pires, L. F., and van Sinderen, M. (2008). Defining and prototyping a life-cycle for dynamic service composition. In *Second International Workshop on Architectures, Concepts and Technologies for Service Oriented Computing (ACT4SOC)*, pages 79–90.

[da Silva et al., 2010] da Silva, E. G., Ferreira Pires, L., and van Sinderen, M. (2010). On the Design of User-Centric Supporting Service Composition Environments. *7th International Conference on Information Technology: New Generations (ITNG)*, pages 666–671.

[Danado and Paternò, 2012] Danado, J. and Paternò, F. (2012). Puzzle : A Visual-Based Environment for End User Development in Touch-Based Mobile Phones. In *Human-Centered Software Engineering*, pages 199–216.

[Daniel et al., 2007] Daniel, F., Yu, J., Benatallah, B., Casati, F., Matera, M., and Saint-Paul, R. (2007). Understanding UI Integration: A Survey of Problems, Technologies, and Opportunities. *IEEE Internet Computing*, 11(3):59–66.

[De Angeli and Battocchi, 2011] De Angeli, A. and Battocchi, A. (2011). Conceptual Design and Evaluation of WIRE: A Wisdom-Aware EUD Tool. Technical Report March.

[Edmond et al., 2005] Edmond, D., O'Sullivan, J., and ter Hofstede, A. (2005). Two main challenges in service description: Web Service tunnel vision and Semantic Myopia. In Bournez, C., editor, *Proceedings of W3C Workshop on Frameworks for Semantics in Web Services*, pages 1–5, Innsbruck, Austria. W3C.

[Fallman, 2003] Fallman, D. (2003). Design-oriented human-computer interaction. *Proceedings of the conference on Human factors in computing systems - CHI '03*, (5):225.

[Firesmith, 2005] Firesmith, D. (2005). Are your requirements complete? *Journal of Object Technology*, 4(1):27–43.

[Fischer and Giaccardi, 2006] Fischer, G. and Giaccardi, E. (2006). Meta-design: A Framework for the Future of End-User Development. In Lieberman, H., Paternò, F., and Wulf, V., editors, *End User Development*, pages 427–457. Springer Netherlands.

[Fischer and Shipman, 2013] Fischer, G. and Shipman, F. (2013). Collaborative design rationale and social creativity in cultures of participation. *Creativity and Rationale*, 7(August):164–187.

[Fischer et al., 2009] Fischer, T., Bakalov, F., and Nauerz, A. (2009). An Overview of Current Approaches to Mashup Generation. *Proceedings of Wissensmanagement*, pages 254–259.

[Geyer, 2000] Geyer, L. (2000). Feature modeling using design spaces. In *Proceedings of the 1st German Workshop on Software Product Lines*, Kaiserslauten, Germany.

[Gilchrist, 1972] Gilchrist, M. (1972). *The psychology of creativity*. Melbourne University Press, Melbourne.

[Goldschmidt, 1991] Goldschmidt, G. (1991). The Dialectics of Sketching. *Creativity Research Journal*, 4(2):123–143.

[Gooch, 2013] Gooch, D. (2013). *Designing Communication Devices for Long Distance Dating Relationships*. PhD thesis, University of Bath.

[Grammel and Storey, 2010] Grammel, L. and Storey, M. (2010). A survey of mashup development environments. In *The smart internet*, pages 137–151.

[Gries, 2004] Gries, M. (2004). Methods for evaluating and covering the design space during early design development. *Integration, the VLSI Journal*, 38(2):131–183.

[Gruber and Russell, 1991] Gruber, T. and Russell, D. (1991). Design Knowledge and Design Rationale: A Framework for Representation, Capture, and Use. Technical report, Knowledge Systems Laboratory, Stanford University.

[Guindon, 1988] Guindon, R. (1988). A Framework for building software development environments: System design as ill-structured problems as an opportunistic process. Technical report, Austin, TX.

[Hazelrigg, 1998] Hazelrigg, G. (1998). A Framework for Decision-Based Engineering Design, Journal. *of Mechanical Design*, (120):653–658.

[Hickey and Kipping, 1996] Hickey, G. and Kipping, C. (1996). Issues in Research. A multi-stage approach to the coding of data from open-ended questions. *Nurse Research*, 4:81–91.

[Hsieh and Shannon, 2005] Hsieh, H.-F. and Shannon, S. E. (2005). Three approaches to qualitative content analysis. *Qualitative health research*, 15(9):1277–88.

[Ibrahim et al., 2010] Ibrahim, N., Frénot, S., and Le Mouël, F. (2010). User-Excentric Service Composition in Pervasive Environments. *24th IEEE International Conference on Advanced Information Networking and Applications (AINA)*, pages 682–689.

[ISO/IEC, 2008] ISO/IEC (2008). ISO/IEC CD 25010: Software engineering - Software product Quality Requirements and Evaluation (SQuaRE) - Software and quality in use models. Technical Report Resolution 937.

[Jonassen, 1997] Jonassen, D. (1997). Instructional design models for well-structured and III-structured problem-solving learning outcomes. *Educational Technology Research and Development*, (1):65–94.

[Jones, 1970] Jones, J. (1970). *Design Methods: Seeds of Human Futures*. Wiley-Interscience, London, New York.

[Jul, 2002] Jul, S. (2002). A framework for locomotional design: toward a generative design theory. *CHI '02 extended abstracts on Human factors in computer systems - CHI '02*, page 862.

[Kang et al., 1990] Kang, K., Cohen, S., Hess, J., Novak, W., and Peterson, A. (1990). Feature-Oriented Domain Analysis: Feasibility Study. Technical Report November, Software Engineering Institute, Carnegie Mellon University, Pittsburgh, Pennsylvania, United States.

[Kapitsaki et al., 2007] Kapitsaki, G., Kateros, D. A., Foukarakis, I. E., Prezerakos, G. N., Kaklamani, D. I., and Venieris, I. S. (2007). Service Composition: State of the art and future challenges. In *Mobile and Wireless Communications Summit, 2007. 16th IST*, pages 1–5.

[Kopecky et al., 2009] Kopecky, J., Vitva, T., and Fensel, D. (2009). MicroWSMO and hRESTS. Technical report, SOA4All consortium.

[Koschmider et al., 2009] Koschmider, A., Torres, V., and Pelechano, V. (2009). Elucidating the mashup hype: Definition, challenges, methodical guide and tools for mashups. In *Proceedings of the 2nd Workshop on Mashups, Enterprise Mashups and Lightweight Composition on the Web at WWW 2009.*, pages 1–9.

[Kriplean et al., 2012] Kriplean, T., Toomim, M., Morgan, J., Borning, A., Ko, A. J., Science, C., and Design, H. C. (2012). Is This What You Meant? Promoting Listening on the Web with Reflect. In *Proceedings of the SIGCHI Conference on Human Factors in Computing Systems*, pages 1559–1568. ACM.

[Krueger, 1992] Krueger, C. (1992). Software reuse. *ACM Computing Surveys (CSUR)*, 24(2):131–183.

[Laga and Bertin, 2012] Laga, N. and Bertin, E. (2012). Widgets and composition mechanism for service creation by ordinary users. *Communications Magazine, IEEE*, (3):52–60.

[Lane, 1990] Lane, T. (1990). *Studying software architecture through design spaces and rules*.

[Lane, 1996] Lane, T. (1996). Guidance for User-Interface Architectures. In Garlan, D. and Shaw, M., editors, *Software Architecture - Perspectives on an Emerging Discipline*, pages 97–115. Prentice-Hall.

[Lieberman et al., 2006] Lieberman, H., Paternó, F., Klann, M., and Wulf, V. (2006). End-User Development: An Emerging Paradigm. In Lieberman, H., Paternò, F., and Wulf, V., editors, *End-User Development*, volume 9, pages 1–8. Springer Netherlands.

[Liu et al., 2012] Liu, L., Liu, X., and Li, X. (2012). Cloud-Based Service Composition Architecture for Internet of Things. In *Internet of Things*, pages 559–564. Springer, Berlin, Germany.

[Lizcano et al., 2008] Lizcano, D., Soriano, J., Reyes, M., and Hierro, J. (2008). EzWeb/-FAST: Reporting on a Successful Mashup-based Solution for Developing and Deploying Composite Applications in the Upcoming 'Ubiquitous SOA'. In *Proceedings of the 10th International Conference on Information Integration and Web-based Applications & Services*, number section 2, pages 15–24. ACM.

[MacLean and McKerlie, 1995] MacLean, A. and McKerlie, D. (1995). Design space analysis and use-representations. In Carroll, J. M., editor, *Scenario-based design: envisioning work and technology in system developmen*, pages 183–207. Wiley, New York.

[MacLean et al., 1991] MacLean, A., Young, R. M., Bellotti, V., and Moran, T. (1991). Questions, options, and criteria: Elements of design space analysis. *Human-Computer interaction*, 6(3-4):201–250.

[Maguire and Bevan, 2002] Maguire, M. and Bevan, N. (2002). User requirements analysis A review of supporting methods. In *Proceedings of IFIP 17th World Computer Congress*, number August, pages 25–30, Montreal, Canada. Kluwer Academic Publishers.

[Martin et al., 2004] Martin, D., Burstein, M., and Hobbs, J. (2004). OWL-S: Semantic markup for web services. Technical report.

[Mehandjiev and De Angeli, 2012] Mehandjiev, N. and De Angeli, A. (2012). End user mashups: analytical framework. In *Proceedings of the 2nd International Workshop on Adaptive Services for the Future Internet and 6th International Workshop on Web APIs and Service Mashups*, pages 36–39, Bertinoro, Italy. ACM.

[Mehandjiev et al., 2010a] Mehandjiev, N., Lecue, F., Wajid, U., and Namoun, A. (2010a). Assisted Service Composition for End Users. In *Proceedings of the 2010 Eighth IEEE European Conference on Web Services*, pages 131–138. IEEE Computer Society.

[Mehandjiev et al., 2014] Mehandjiev, N., Namoun, A., and Lécué, F. (2014). End Users Developing Mashups. In Bouguettaya, A., Sheng, Q. Z., and Daniel, F., editors, *Web Services Foundations*, pages 709–736. Springer New York, New York, NY.

[Mehandjiev et al., 2010b] Mehandjiev, N., Namoune, A., Wajid, U., Macaulay, L., and Sutcliffe, A. (2010b). End User Service Composition: Perceptions and Requirements. In *Web Services (ECOWS), 2010 IEEE 8th European Conference on*, pages 139–146.

[Mennie and Pagurek, 2000] Mennie, D. and Pagurek, B. (2000). *An Architecture to Support Dynamic Composition of Service Components and its Applicability to Internet Security*. PhD thesis, Ottowa.

[Minhas et al., 2012] Minhas, S. S., Sampaio, P., and Mehandjiev, N. (2012). A Framework for the Evaluation of Mashup Tools. In *2012 IEEE 9th Intl. Conf. on Services Computing*, pages 431–438. IEEE.

[Mittal, 2010] Mittal, K. (2010). Service-Oriented Unified Process (SOUP). http://www.kunalmittal.com/html/soup.html. last accessed 28th Agust 2014.

[Na et al., 2010] Na, L., Patel, A., Latih, R., Wills, C., Bangi, U. K. M., and Ehsan, S. D. (2010). A Study of Mashup as a Software Application Development Technique with Examples from an End-User Programming Perspective. *Jounal of Computer Science*, 6(April):1406–1415.

[Namoun et al., 2010a] Namoun, A., Nestler, T., and De Angeli, A. (2010a). Conceptual and Usability Issues in the Composable Web of Software Services. In Daniel, F. and Facca, F., editors, *Current Trends in Web Engineering*, volume 6385, pages 396–407. Springer Berlin / Heidelberg.

[Namoun et al., 2010b] Namoun, A., Nestler, T., and De Angeli, A. (2010b). End User Requirements for the Composable Web. In *Second Intl. Workshop on Lightweight Integration of the Web at ICWE 2010*, Vienna, Austria.

[Namoun et al., 2010c] Namoun, A., Nestler, T., and De Angeli, A. (2010c). Service Composition for Non-programmers: Prospects, Problems, and Design Recommendations. In *Web Services (ECOWS), 2010 IEEE 8th European Conference on*, pages 123–130.

[Namoun et al., 2010d] Namoun, A., Wajid, U., and Mehandjiev, N. (2010d). Service Composition for Everyone: A Study of Risks and Benefits. In Dan, A., Gittler, F., and Toumani, F., editors, *Service-Oriented Computing. ICSOC/ServiceWave 2009 Workshops*, volume 6275, pages 550–559. Springer Berlin / Heidelberg.

[Nelson and Stolterman, 2003] Nelson, H. and Stolterman, E. (2003). *The design way: Intentional change in an unpredictable world: Foundations and fundamentals of design competence*. Educational Technology Publications.

[Nestler et al., 2011] Nestler, T., Namoun, A., and Schill, A. (2011). End-user development of service-based interactive web applications at the presentation layer. In *Proceedings of the 3rd ACM SIGCHI symposium on Engineering interactive computing systems*, pages 197–206, Pisa, Italy. ACM.

[Nowak and Pautasso, 2011] Nowak, M. and Pautasso, C. (2011). Goals , Questions and Metrics for Architectural Decision Models. In *Proceedings of the 6th International Workshop on SHAring and Reusing Architectural Knowledge.* ACM.

[Obrenović and Gašević, 2008] Obrenović, v. and Gašević, D. (2008). End-user service computing: spreadsheets as a service composition tool. *IEEE Transactions on Services Computing*, 1(4):229–242.

[Olewnik, 2005] Olewnik, a. T. (2005). On Validating Engineering Design Decision Support Tools. *Concurrent Engineering*, 13(2):111–122.

[Osborne, 1953] Osborne, A. (1953). Applied imagination: principles and procedures of creative problem solving. *Charles Scribener's Sons, New York*.

[Owen, 1997] Owen, C. L. (1997). Building the knowledge base. *Journal of the Japanese Society for the Science of Design*, 5(2):36–45.

[Papazoglou and Georgakopoulos, 2003] Papazoglou, M. P. and Georgakopoulos, D. (2003). Service -Oriented Computing: Concepts, Characteristics and Directions. In *Proceedings of the Fourth International Conference on Web Information Systems Engineering (WISE)*, volume 0, pages 3–12. IEEE.

[Parker et al., 2012] Parker, A., Kantroo, V., Lee, H., and Osornio, M. (2012). Health promotion as activism: building community capacity to effect social change. In *Proceedings of the SIGCHI Conference on Human Factors in Computing Systems*, pages 99–108. ACM.

[Peltz, 2003] Peltz, C. (2003). Web services orchestration and choreography. *Computer*, 36(10):46–52.

[Perlis, 1982] Perlis, A. (1982). Epigrams on programming. *SIgPLAN Notices*, 17(9).

[Picozzi, 2010] Picozzi, M. (2010). *DashMash: a mashup environment for end user development*. Master's thesis.

[Picozzi, 2014] Picozzi, M. (2014). *End-user Development of Mashups: Models, Composition Paradigms and Tools*. PhD thesis, Politecnico Milano.

[Pierce, 2012] Pierce, J. (2012). Undesigning Technology : Considering the Negation of Design by Design. In *Proceedings of the SIGCHI Conference on Human Factors in Computing Systems*, pages 957–966. ACM.

[Pietschmann et al., 2010] Pietschmann, S., Nestler, T., and Daniel, F. (2010). Application composition at the presentation layer: alternatives and open issues. In *Proceedings of the 12th International Conference on Information Integration and Web-based Applications & Services*, pages 461–468. ACM.

[Pietschmann et al., 2009] Pietschmann, S., Voigt, M., Rümpel, A., and Meiß ner, K. (2009). Cruise: Composition of rich user interface services. *Web Engineering*, 5648:473–476.

[Pinelle et al., 2003] Pinelle, D., Gutwin, C., and Greenberg, S. (2003). Task analysis for groupware usability evaluation: Modeling shared-workspace tasks with the mechanics of collaboration. *ACM Transactions on Computer-Human Interaction (TOCHI)*, 10(4):281–311.

[Rabiee, 2007] Rabiee, F. (2007). Focus-group interview and data analysis. *Proceedings of the Nutrition Society*, 63(04):655–660.

[Ramollari et al., 2007] Ramollari, E., Dranidis, D., and Simons, A. (2007). A survey of service oriented development methodologies. *The 2nd European Young Researchers Workshop on Service Oriented Computing*, page 75.

[Rao and Su, 2005] Rao, J. and Su, X. (2005). A Survey of Automated Web Service Composition Methods. In Cardoso, J. and Sheth, A., editors, *Semantic Web Services and Web Process Composition*, volume 3387, pages 43–54. Springer Berlin / Heidelberg.

[Rasmussen et al., 1994] Rasmussen, J., Pejtersen, A. M., and Goodstein, L. P. (1994). *Cognitive Systems Engineering*. Wiley, New York.

[Ridge and O'Neill, 2014] Ridge, A. and O'Neill, E. (2014). Establishing requirements for End-user Service Composition tools. *Requirements Engineering*, pages 1–29.

[Rieman, 1993] Rieman, J. (1993). The diary study: a workplace-oriented research tool to guide laboratory efforts. In *Proceedings of the INTERACT'93 and CHI'93 conference on Human factors in computing systems*, pages 321–326. ACM.

[Rittel, 1984] Rittel, H. (1984). Second-Generation Design Methods. In *Developments in design methodology*, pages 317–327. Wiley.

[Ro et al., 2008] Ro, A., Xia, L., Paik, H.-Y., and Chon, C. (2008). Bill Organiser Portal: A Case Study on End-User Composition. In Hartmann, S., Zhou, X., and Kirchberg, M., editors, *Web Information Systems Engineering: WISE 2008 Workshops*, volume 5176, pages 152–161. Springer Berlin / Heidelberg.

[Rodríguez-Mier et al., 2010] Rodríguez-Mier, P., Mucientes, M., Lama, M., and Couto, M. (2010). Composition of web services through genetic programming. *Evolutionary Intelligence*, 3(3):171–186.

[Rogers et al., 2011] Rogers, Y., Sharp, H., and Preece, J. (2011). *Interaction Design*. Wiley.

[Rosson and Kellogg, 1987] Rosson, M. B. and Kellogg, W. A. (1987). 1987 Designing for Designers : An Analysis of Design Practice in the Real World GI 1987 The Design Population. pages 137–142.

[Rubio et al., 2004] Rubio, S., Díaz, E., Martín, J., and Puente, J. (2004). Evaluation of Subjective Mental Workload: A Comparison of SWAT, NASA-TLX, and Workload Profile Methods. *Applied Psychology*, 53(1):61–86.

[Schön, 1983] Schön, D. A. (1983). *The Reflective Practitioner: How Professionals Think in Action*. Basic Books, New York.

[Schön, 1987] Schön, D. A. (1987). *Educating the reflective practitioner*. Jossey-Bass, San Francisco.

[Shneiderman, 2000] Shneiderman, B. (2000). Creating creativity: user interfaces for supporting innovation. *ACM Transactions on Computer-Human Interaction (TOCHI)*, 7(1):114–138.

[Shupe et al., 1988] Shupe, J., Muster, D., Allen, J., and Mistree, F. (1988). Decision-Based Design: Some Concepts and Research Issues. In Kusiak, A., editor, *Expert Systems, Strategies and Solutions in Manufacturing Design and Planning*, pages 3–37. Dearborn, MI.

[Simon, 1996] Simon, H. A. (1996). *The Sciences of the Artificial (3rd Ed.)*. MIT Press, Cambridge, MA.

[Sinderen, 1995] Sinderen, M. J. (1995). Design Quality Criteria. In *On the design of application protocols.*, pages 19–38.

[Sommerville, 2011] Sommerville, I. (2011). *Software Engineering*. Pearson/Addison Welsey, Harlow, England, New York, 9 edition.

[Stecca and Maresca, 2010] Stecca, M. and Maresca, M. (2010). Mashup Patterns from Service Component Taxonomy. In *Advanced Information Networking and Applications Workshops (WAINA), 2010 IEEE 24th International Conference on*, pages 12–17.

[Stevens et al., 2006] Stevens, G., Quaisser, G., and Klann, M. (2006). Breaking It Up: An Industrial Case Study of Component-Based Tailorable Software Design. In Lieberman, H., Paternò, F., and Wulf, V., editors, *End User Development*, pages 269–294. Springer Netherlands.

[Sutcliffe, 1998] Sutcliffe, A. (1998). Scenario-based requirements analysis. *Requirements Engineering*, 3(1):48–65.

[Sutcliffe, 2003] Sutcliffe, A. (2003). Scenario-based requirements engineering. In *Requirements Engineering Conference, 2003. Proc. 11th IEEE Intl.*, pages 320–329.

[Sutcliffe et al., 1998] Sutcliffe, A., Maiden, N., Minocha, S., and Manuel, D. (1998). Supporting scenario-based requirements engineering. *Software Engineering, IEEE Transactions on*, 24(12):1072–1088.

[Sutcliffe and Ryan, 1998] Sutcliffe, A. and Ryan, M. (1998). Experience with SCRAM, a scenario requirements analysis method. In *Requirements Engineering, 1998. Proceedings. Third International Conference on*, pages 164–171. IEEE.

[Szafir and Mutlu, 2012] Szafir, D. and Mutlu, B. (2012). Pay attention!: designing adaptive agents that monitor and improve user engagement. In *Proceedings of the SIGCHI Conference on Human Factors in Computing Systems*, pages 11–20. ACM.

[Thurston, 2001] Thurston, D. L. (2001). Real and Misconceived Limitations to Decision Based Design With Utility Analysis. *Journal of Mechanical Design*, 123(2):176.

[Tosic et al., 2000] Tosic, V., Mennie, D., and Pagurek, B. (2000). Dynamic Service Composition and Its Applicability to E-Business Software Systems - The ICARIS Experience. In *Workshop on OO Business Sol ECOOP*, Budapest, Hungary.

[Tuchinda et al., 2011] Tuchinda, R., Knoblock, C. a., and Szekely, P. (2011). Building Mashups by Demonstration. *ACM Transactions on the Web*, 5(3):1–45.

[Vulcu et al., 2008] Vulcu, G., Bhiri, S., Hauswirth, M., and Zhou, Z. (2008). A User-Centric Service Composition Approach. In Meersman, R., Tari, Z., and Herrero, P., editors, *On the Move to Meaningful Internet Systems: OTM 2008 Workshops*, volume 5333, pages 160–169. Springer Berlin / Heidelberg.

[W3C, 2001] W3C (2001). WSDL 1.1: Web Services Description Language 1.1. Technical Report 6th August.

[W3C, 2004a] W3C (2004a). OWL-S: Semantic Markup for Web Services Version 1.0. http://www.w3.org/Submission/OWL-S/.

[W3C, 2004b] W3C (2004b). WS-CDL: Web Services Choreography Description Language 1.0. http://www.w3.org/TR/2004/WD-ws-cdl-10-20041012/.

[W3C, 2005a] W3C (2005a). WSDL-S: Web Service Semantics. http://www.w3.org/Submission/WSDL-S/.

[W3C, 2005b] W3C (2005b). WSMO: Web Services Modelling Ontology. http://www.w3.org/Submission/WSMO/.

[W3C, 2007] W3C (2007). SAWSDL: Semantic Annotations for WSDL and XML Schema. http://www.w3.org/TR/sawsdl/.

[W3C, 2010] W3C (2010). WSMO-Lite: Lightweight Semantic Descriptions for Services on the Web. http://www.w3.org/Submission/2010/SUBM-WSMO-Lite-20100823/.

[Wagner and Deissenboeck, 2008] Wagner, S. and Deissenboeck, F. (2008). Abstractness, Specificity, and Complexity in Software Design Categories and Subject Descriptors. In *Proceedings of the 2nd International Workshop on the Role of Abstraction in Software Engineering*, pages 35–42. ACM.

[Wajid et al., 2010] Wajid, U., Namoune, A., and Mehandjiev, N. (2010). A comparison of three service composition approaches for end users. In *Proc. Intl. Conf. on Advanced Visual Interfaces*, page 407, Roma, Italy. ACM.

[Wallas, 1926] Wallas, G. (1926). The art of thought.

[Warr, 2007] Warr, A. (2007). *Understanding and supporting creativity in design*. PhD thesis, University of Bath.

[Westerlund, 2005] Westerlund, B. (2005). Design space conceptual tool: grasping the design process. In *Proceedings for â€˜In the Makingâ€™, Nordes, the Nordic Design Research Conference*.

[Westfall, 2009] Westfall, L. (2009). *The certified software quality engineer handbook*. ASQ Quality Press.

[Wharton, 1992] Wharton, C. (1992). Cognitive Walkthroughs: Instructions, Forms and Examples. Technical report.

[Wood and Agogino, 2005] Wood, W. H. and Agogino, A. M. (2005). Decision-Based Conceptual Design: Modeling and Navigating Heterogeneous Design Spaces. *Journal of Mechanical Design*, 127(1):2.

[Yu et al., 2008] Yu, J., Benatallah, B., Casati, F., and Daniel, F. (2008). Understanding Mashup Development. *IEEE Internet Computing*, 12(5):44–52.

[Zhang et al., 2003] Zhang, R., Arpinar, I. B., and Aleman-Meza, B. (2003). Automatic composition of semantic web services. *Automatic Composition of Semantic Web Services*, 3:38–41.

[Zowghi and Gervasi, 2003] Zowghi, D. and Gervasi, V. (2003). On the interplay between consistency, completeness, and correctness in requirements evolution. *Information and Software Technology*, 45(14):993–1009.

APPENDIX

A Requirements Study Materials

A.1 Consent Form

Study Overview

This study aims to assess how assistance can be provided in creating the conceptual design for a Service Composition tool/application. To do this, we will be asking you to create a conceptual design for a Service Composition application, given a particular type of help. The session should take no more than an hour.

During the session, you will be introduced to Service Composition, as well as being shown what we want you to create [15 minutes]. We'll also present you with a questionnaire to see what experience you've had with Service Composition (or similar concepts) before [5 minutes]. The main stage of the session will be to create a conceptual design for a Service Composition tool/application [max 30 minutes]. Finally we'll give you another questionnaire to ask how you think the session went [10 minutes]. The session will be recorded on video and then the audio from the session will be transcribed anonymously in order to find any problems that you had during the session. During this process, the data will be stored securely.

Important Information

- All data collected during this study will be recorded such that your individual results are anonymous and cannot be traced back to you.
- Your results will not be passed to any third party and are not being collected for commercial reasons.
- Participation in this study does not involve physical or mental risks outside of those encountered in everyday life.
- All procedures and information can be taken at face value and no deception is involved.
- You have the right to withdraw from the study at any time and to have any data about you destroyed. If you do decide to withdraw, please inform the experimenter.

By signing this form you acknowledge that you have read the information given above and understand the terms and conditions of this study.

Experimenter: Andy Ridge, Dept. of Computer Science. A.D.Ridge@bath.ac.uk

Name:			
Age:		Sex:	
Occupation			
Signed:		Date:	

A.2 Participant Briefing

If you are unclear about any of the terminology used in this document, please either refer to the glossary or ask the experimenter.

Session Briefing

The aim of this session is to gather and validate requirements for end-user Service composition, which we will be testing using a prototype mobile phone application which will allow us to compose services. In the session, you will first be introduced to the concepts involved in service composition, which will be clarified with some scenarios to tell you how service composition could be used. After this, you will be shown our prototype Service Composition application. The rest of the session will involve us working through the application with a script to gather information about how you might want to use such a tool. We will also compare the features of our tool with other available Service Composition tools.

The format of sessions will be as follows:

1. We will outline a feature of our prototype composition application, explaining what is involved in the process, and what you get out of it.
2. We will then show this same feature in other composition tools, and try to highlight the differences between the approaches (if they aren't obvious)
3. You will then be asked a series of questions about the feature, with prompt if you are unsure.
4. This will then be repeated for other features (there are three features in total)

Potential users of this tool should be people who are currently familiar with (and regularly use) their smartphone, and frequently download and use mobile apps. We aren't restricting our potential users based upon the platform on which their smartphone operates (although the prototype relies on use of Android). During the session, we are looking to gather requirements from the elements of the platform that you feel are the most important to you, as a potential user of such a system. Note that you are not required to be familiar with Android in order to participate in this study, and if you have any questions as to how it is different from your phone then please ask.

In particular, we're interested in your opinions of the general concepts of service composition, as well as some of the design decisions that have been made in making the prototype (these will be highlighted to you during the session). However, we are not interested in specific interface problems (buttons being in the wrong place, colours, etc.)

Introduction: "There's almost an app for that"

For our purposes, a service can be thought of as a function that is carried out by a piece of software. Examples of this could be (the software that allows) looking up a contact on your phone, posting a Facebook status update, or doing a Google search. Mobile apps tend to be collections of such services (which can be connected together, or not connected) that the developer making the app has gathered together so that it can be more useful for you, the user.

Service composition allows users to combine these individual services together in different ways so that they can make something that is more useful for them, so that they can build their own apps/services that do exactly what they want, rather than what the app developer thinks they want. In our system, developers still make the components (which can be bundled inside their apps), but the end-user is the one who decides how to combine these components to create a new service.

Currently, Service Composition uses technical terminology that most end-users would probably not be familiar with, which is something we do wish to investigate as part of this work (including the term "Service Composition" itself). However, this is not the aim of this session, meaning that we will need to provide some definitions of the technical terms that you will encounter during the session. These terms can be seen in the Glossary, which you should look at now (and at any other point throughout the session).

A.3 Glossary

Service: A service is a task that is performed by some software to achieve a goal.

Component (Atomic Service): An atomic service (or component/component service) is a single service that can be used in composition, which cannot be broken down into smaller components.
Example atomic/component services: looking up a bus timetable, converting from Fahrenheit to Celsius, converting from USD to GBP, posting a Tweet to Twitter, posting a Facebook status.

Composite Service: The output of the service composition process – A coordination of atomic/component services that have been arranged and linked together to create a new functionality.
Examples composite services: Posting a photo to Facebook – (uploading a photo, listing friends who are to be tagged in the photo, adding location, posting the photo to your profile). Price comparison service (lookup price on one website, lookup price on other website, compare prices).

Service Composition: The process of coordinating or connecting a group of components together to create a composite service.

Inputs: The data that needs to be passed to a component (from another component) in order for it to execute.

Outputs: The data that a component will return once it has executed (which can be passed to another component).

User Inputs (Parameters): Customisable parameters that the user can set before they run the service. For example, in a Facebook service, you would need to provide your username and password before you could post a status update, but you wouldn't need to do this to do a Google Search.

A.4 Scenarios

Scenario character: Ben

Ben lives in greater London with his girlfriend, and works in central London as a consultant. He commutes to work in the morning using the tube, which normally takes him about an hour. He often has meetings outside the office, most of which are in London, but are occasionally in other parts of the country, where he would normally drive.

He is a smartphone owner, and likes to have an up-to-date phone, so upgrades often. His current phone is an Android smartphone, which he tends to prefer, but has also owned smartphones on other platforms.

Scenario 1: Tube Problem Notifications

Ben has a tube status service for his smartphone that allows him to look up the status of any tube line. He feels that it's too much hassle to check each of these manually and wants his phone to notify him when there is a problem using the in-built notification service in the phone. There's no option in the service itself to do this, so he decides to use End-User Service Composition to fix his problem. After choosing to edit the service, he is able to compose the phone's notification service onto the event-based part of the tube service, so that when a problem is reported on a particular line (which he can then select), he will be notified via a new item in the notification tray and phone's default notification tone will sound to alert him.

After using this new service for a few days and getting notified at strange times of day he decides that he only wants to receive these notifications around the times he would normally be getting the tube. So he again chooses to edit the service, but this time adds the device's clock service in between the tube service and the notification service. He sets the clock to only let the notifications through between 6-8am, and 4-7pm.

Scenario 2: Meeting Delay Notifications

Ben has regular meetings across central London with other companies that he has dealings with in his role. He schedules these meetings in his calendar, and invites the other attendees so that he has access to their contact details. Normally he would use the tube to get to these meetings, but because of the preparation for the Olympics, there have been delays and it's affected him getting to the meetings, and he has had to send separate messages to each of the meeting's attendees to let them know.

He composes the tube lookup service with the calendar, so that it will check for tube problems an hour before each of his meetings. He then composes these with an email service to allow it to send messages to other attendees. If the tube lookup finds a problem on a line that he needs to use to get there, it sends an email to the other attendees to inform them that Ben might be late.

Scenario 3: SMS Trigger

Ben is driving back to London after having a meeting in York earlier in the day. He looks up the traffic on Google Maps before he leaves York and sees that the traffic is already quite heavy on the route he wants to take. He's planning on going out with his girlfriend in the evening, so rings her to let her know that he might be late. He wants to be able to keep her informed of his progress down the motorway, but knows that he can't call her while he's driving.

Ben creates a service that his girlfriend can use that will send her a text containing his current location. He composes a SMS receiving service, and sets it to respond to his girlfriend's number when she sends him a SMS saying "where?". He then adds a location service and a SMS sending service to the composition, and sets the SMS to go to his girlfriend. This means that when his girlfriend sends him a text which contains "Where?", his phone will automatically reply to her with his current location, so that he doesn't have to.

Scenario 4: Morning Weather Notifications

Ben has a walk every morning to get from the tube station to his office, and back again. He knows he should look up the weather when he leaves the house in the morning, but normally only takes an umbrella if it's raining when he's leaving. To fix this problem, he composes a weather lookup service with a notification service, which informs him of the day's weather. He sets this service to run in the morning when he gets up, so that he knows about the day's weather before he leaves the house.

A.5 Participant Questionnaire

Demographics

1. How old are you?

2. What is your sex? (Please choose *one* answer)

. Male
. Female

3. What is the highest degree or level of education you have completed? If currently enrolled please indicate the highest you have attained previously. (Please circle *one* answer)

. None
. GCSEs or equivalent
. A/AS or equivalent
. BSc/BA or equivalent
. MSC/MA or equivalent
. PhD or equivalent

4. In what field is your highest qualification?

5. What is your current employment status?

. Unemployed
. Self-employed
. Employed
. Student
. Retired
. Unable to work

Technical Expertise

1. Do you own a smartphone? (Please choose *one* answer)

. Yes
. No

2. If so, what platform does it run? (Please circle *one* answer)

. Android
. iOS (iPhone)
. Windows Phone
. Blackberry OS
. Other:

3. What make and model is your smartphone?

4. Roughly how many apps do you download? (Please circle *one* answer)

. Daily
. Weekly
. Monthly
. Less than monthly

5. What type of apps do you download most often? (Please circle at least *one* answer)

. Games
. Utilities
. Social
. Other:

6. What was the last app you downloaded? Do you remember why you downloaded it?

7. In what ways do you normally find out about apps? (Please circle at least *one* answer)

. Recommendations from friends
. Recommendations from the internet
. Searching
. Browsing

8. If you were searching for an app (not necessarily a specific app), what attribute or feature would you normally use to search for it?

9. Have you ever done any smartphone development? (Please circle *one* answer)

. Yes
. No

10. If so, on which platform(s)? (Please circle at least *one* answer)

. Android
. iOS (iPhone)
. Windows Phone
. Blackberry OS
. Other:

11. Have you ever released any mobile apps onto a mobile app store? (Please circle *one* answer)

. Yes
. No

12. If so, on which platform(s)? (Please circle at least *one* answer)

. Android ☑
. iOS (iPhone) ☑
. Windows Phone ☑
. Blackberry OS ☑
. Other: ☑

13. How many?

Service Composition

1. Have you ever used a Service Composition application before? (Please circle *one* answer)

. Yes ☑
. No ☑
. Not sire ☑

2. If yes, what is the name of that tool?

Domain Knowledge

1. How often do you travel using the tube? (Please circle at least *one* answer)

. Daily ☑
. Weekly ☑
. Monthly ☑
. Less than monthly ☑

2. If you use the tube frequently, do you normally check the status of the lines you need before travelling (Please circle *one* answer)

. Yes ☑
. No ☑
. Not sure ☑

3. Do you use weather services on your mobile phone? (Please circle *one* answer)

. Yes ☑
. No ☑

4. Have you ever used a system or app that sends texts for you automatically (Please circle *one* answer)

. Yes ☑
. No ☑

A.6 Requirements Study Demonstrator Script

Viewing Available Components

Aim for participants: To view the atomic services that they can make new composite services out of, to get an idea of what can be built using these components.

Aim for researcher: To assess how potential users view atomic services that they can use to build composite services.

View the list of Components At the bottom of the home page, you'll see a button labelled "Components", tap that button and you'll be taken to a list of the atomic services that you can make composite services out of.

When you get to this page, you see two lists of services:

1. "Device": Services that are on the device
2. "Web": Services that aren't currently on the device, and need to be downloaded from the Web.

If you tap each of these tabs, you will see two different lists of atomic services, with a small amount of information presented about each service. If you tap on one of the services, you will be able to find out more information about the service.

View more information about one of the component services Tapping one of the services brings up a page where you can find out more information about the service. You can see more information regarding its functionality, how it connects with other services, and any parameters that you can set.

Task flow for other applications

Tasker

1. Navigate to pre-made service and tap add service, then show them the list of categories and a few sub-lists of services.
2. Clicking on a service brings up the parameter setting page, so just tell them you can't get more info than what is in the list and be done with it.

On{X}

1. Go to the documentation page and show them the list of services down the left hand side.
2. Show them the main content and point out that this is the same as the page we had where we were describing the functionality of the service.

IFTTT

1. Click the channels link and show them the list of service providers. Point out the ones in colour are the ones that I've already set up. Tap on a service provider (Dropbox?) and show them the different services available, as well as the other stuff on the page.
2. Click on one of the services and show them the list of ingredients at the bottom, where the ingredients are the things that can be passed to other services.

Yahoo! Pipes

1. Point out the list on the left hand side of the page, click one of the categories to show the available services inside it.
2. Point out the description underneath, and click on the learn more link to see more information about the service.

Automator

1. Point out the list of service providers and services on the left hand side. Show that clicking one of the providers filters the list of available services.
2. Show that clicking on one of the services changes the pane at the bottom, which shows more information about each of the services.

Quartz Composer

1. Show them the patch library list of services
2. Show them the pane underneath that provides information about each patch

Questions and probes

1. What might you want to know about a service in the list of components?
   - What attributes should it present to you?
   - What do you need to know about an app to decide whether to download it or not?
2. How should services in the list of components be grouped?
   - By the app that provides them?
   - By category?
3. How should services be displayed?
   - As if they are apps on a phone – in a grid
   - As if they are apps in an app store – in a list
4. How would you want to find services?
   - Search by name, features, etc.
   - Browse by category, etc.
   - Browse for or search by the app that they are in
5. What might you want to know about an individual service, once you've selected it from the list?
   - What would make you choose this service over another?

Create your own composition (Tube & Notification Services)

Aim for participants: Create a service that they can use later using service composition, (mostly) following what Ben does in the first scenario.

*Aim for researcher:*To see if users can understand the composition process as a whole; to identify the aspects of the composition that they may not understand, and to see if we can improve their understanding.

Create a Composite Service You should currently be on a page showing you either a single atomic service, or a list of composite services. You can either use the back button or the home button to get back to the "My Services" page. From this page, you can then tap "New Service" to get to the composition page. From this page we can create the composition. Note: This is the process that Ben would have to go through in our first scenario.

Add the Tube Service to the composition .

To add a new service to the composition, you can either tap the empty canvas or tap the "Add Component" button at the bottom of the screen. Tapping this will take you to lists that look very much like those we saw in the last section. The main difference here is that the services that are compatible with your current composition are highlighted for you.

In this case, you don't have any services in your composition yet, so the services that don't have any inputs are highlighted. To clarify, this means that the service doesn't need any information to be passed to it from *other services*.

Tap the Tube Service (which should be highlighted) to bring up more information about the Tube Service. Once you've read through the features of the service, tap the tick button at the top of the page.

The tube service will now be displayed in the composition. Underneath the Tube Service, you will also see a yellow bar, which indicates that there could be a problem with the composition. If you tap the yellow bar, you will see the reason for the warning. In this case, the warning is being displayed because the Tube Service gets some information and gives it back to the Composer app, but then nothing happens with this information, it is lost.

Add the Notification Service to the composition. Tap the add button again (this time you have to use the button rather than tapping the canvas). This time the services that are highlighted are the services that have inputs (data from services) that are compatible with the output of the Tube Service. Tap the Notification Service (which should be highlighted) and tap the tick to add it to the composition.

Now that the Notification Service has been added to the composition, you can see it underneath the Tube Service, and the yellow warning bar has disappeared because you're now doing something with the information that is passed out of the Tube Service.

Test the composition you've made One of the buttons at the bottom of the page gives you the option to test the composite service that you've made. Tap the test button now.

When you try to test the service, you'll be asked about setting the parameters to the service. This is because the Tube service is customisable so that you can set the tube lines that you're interested in finding out about. For now, say that you don't want to set any parameters to the service, which means you'll see the problems on all the tube lines.

Once the service has executed, you should see one or more notifications in the notifications tray that will tell you about any problems on the tube. On this version, you will also see a notification if everything is okay (though obviously this wouldn't be there in the final version).

Test the service again, but this time try setting the parameters to the service.

Save your composition Click the save button at the top of the page, and you will be taken to the page where you can enter information about the service. Give your service a name, and write a short description. It's important to make these useful so that you'll be able to tell what the service does later. Click the save button at the bottom of the page, and if you've entered everything correctly, you'll be taken back to the composition page.

Create a composition with a Trigger Service Now we need to create another service, but this time we need to use the "Received SMS service", which is a trigger-based service. That means that the service is not run by you when you want, it activates when some external event occurs (in this case when you receive a SMS).

If you add this service to the composition, followed by adding the on-screen message service, you will have a completed composition. If you try to test the service, you will be informed that because it has a trigger, you can't test the service. Instead, save the service. Now if you go to the messaging app and send yourself a text message, you will see that your service activates, and the message is shown on screen.

Task flow for other applications

Note that due to the differences in what is capable with these applications (and available services within them), we can't create exactly the same service in each application

Tasker: Compose a service to flash text if battery level is below 50

1. Create new task called "Battery status"
2. Add service: Alert → Flash
3. Set text to "You've got %BATT% battery"

On{X}: Compose a service to show Google when you unlock your phone

1. Click "create"
2. Enter:
```
device.screen.on(''unlock'', function(){
device.browser.launch(''http://www.google.co.uk'');
console.log('Hello World notification was sent to the phone')
});
```
3. Click "save and send to phone"

IFTTT: Compose a service to email me if it's going to rain tomorrow

1. Create a new recipe
2. Click “this”, “Weather”, “tomorrow’s forecast calls for”, and “rain”
3. Click “that”, “Email”, “Send me an email”

Yahoo! Pipes: Compose a service which filters an RSS feed to only contain information about a certain subject: e.g. “Syria”

1. Click “new pipe”
2. Drag the “Fetch feed” module into the thing and enter the URL of the BBC News feed: “http://feeds.bbci.co.uk/news/rss.xml”
3. Connect it to the “Pipe Output”
4. Look for a word in the subject that we’re interested in
5. Add in the “Filter” module, chance the rule to “Permit”, “item.title” and the subject.
6. Link all the stuff together

Automator: Rename some items in Finder to add the date/time to the end of the filename.

1. Add “Find Finder items” to the workflow from “Files & Folders”, and enter some filename to search for
2. Add “Copy finder items” to the workflow
3. Add “Rename Finder items” to the workflow, and set it to add date or time.

Quartz Composer: Create something where round, red particles follow the mouse

1. Create a blank composition
2. Add “Clear” to the pane to clear the background
3. Add “Particle system” to the composition. Click “parameters” and set “Color” to red.
4. Add “Lenticular halo” to the composition. Link the “Image” output to the “Image” input of “Particle System”
5. Add “Mouse” to the composition. Link the outputs “X Position” and “Y Position” to the relative inputs of “Particle System”

Questions / Probes

1. What flow do you think is going on in THIS composite service?
   - Describe what you think is going on in the composition – how are the services connected to one another.
     - This **THEN**this
     - Passes **DATA TYPE**to this (data type is important, if they just say data then they don’t mean this one).
2. What do you want to know about each service in the composition?
   - You need to be able to identify the services in the composition, and work out what they are doing
3. When you’re making the composition, do you care what the inputs and outputs to the services are?
   - Or is the fact that they are compatible the only important thing?
   - The information that could be provided at the moment is the name of the input or output, a description and a type.
4. How should connections between services be represented?
   - Explicitly – box and lines
   - Implicitly in a list – they are next to each other, so a connection is implied

5. How do you want to know which services are compatible with the ones you've got so far?
    - Would you want to see them all and highlight the good ones?
    - Only show the good ones?
    - No help?
6. How much help do you need in designing compositions?
    - Templates from other people
    - Use and edit other people's shared compositions
    - Sample compositions
7. Do you think you'd need to test compositions?
    - What about services that happen at a particular time or they respond to an action?
8. Does the metaphor employed by other tools make them more intuitive
    - Plumbing – Yahoo! Pipes
    - Logic – IFTTT and On{X}
    - Cooking – IFTTT and On{X}
9. Is it important that a trigger service is different from the others?
    - Would you want to know when you are choosing it that it's different from the others?
10. Is it made more difficult that you can't test it?
    - What might you do to try to work out if the service does what you want if you can't test it?

View, Run, Edit the Service from the Home page

*Aim for participant:*Assess how a user can use and edit the services that they've created.

*Aim for Researcher:*Identify how possible users might want to interact with composite services once they've created them.

View the home (My Services) page and interact with a service This is the screen where you'll be able to see all of the composite services that you create using the app. At the moment you can see sample services that we've created for you. If you tap on a service on this page, you get a list of interactions that you can have with that service.

Tap "Tube Service" (or similar) now, and choose to run it. This is a service which outputs issues on the Tube to the built-in Notification system, so when you run the service you will either see notifications for failures on any tube lines, or a different notification if there are no problems found on the tube.

Run your saved service from the home page, and set the parameters If you aren't there already, go back to the home page, either by clicking the back button on the device, or the back arrow at the top of the page. You should see the service you've created, which you can interact with by tapping on it. Tap on the service and select run to run the service again. Because this is the first time you've run the service properly (i.e. not been testing it), you should be asked if you want to set the parameters for the service, which will be saved for the next time you run the service.

Note that you can edit these parameters later, by clicking the parameter option at the top of the page.

Run the service on a timer so that it goes off once per minute Another interaction you can have with the service is to set it to run on a timer. Tap your service and select this option now.

This will display a dialog that gives you the option of how often to run your service, choose 1 minute, and tick the "run service now" checkbox. When you press okay, the service will execute, and will then run every minute from then on (currently until you restart the device).

Stop the service from running on a timer Since the service will now run forever, you should be able to stop the service. Tap the option to view running services at the bottom of the home page ("Running services"). The composite services that are running on the device are shown, which should currently only show the composite service that you created.

There are two interactions you can have with the running service, either pausing it or stopping it. If you pause the service, it stays in the list and can be resumed at any time. If you stop the service, it will not run any longer and be removed from the running services list. Stop your service now.

Task flow for other applications

Tasker

1. Tap the profiles tab, and tap the plus to create a new one
2. Name it something, and choose "state", then "hardware", then "USB Plugged"
3. Then in the task selection, choose "Battery State"

On{X}

1. Show the composite service list on the mobile app, point out the text stuff
2. Show the composite service on the website, and then click on one of the services to do parameters – can't be the one we made earlier because apparently you can't change parameters of things that aren't recipes. Change the parameters of a service.
3. Show that you can duplicate a service based on its code
4. Show that you can get other services from already available recipes:
   - Click "Recipes"
   - Click on one of the services
   - Set its parameters
   - Click "add"

IFTTT

1. Show "My Recipes"
   - Point out what the services look like – as they did in the composition, and show the different interactions that you can have with the services
   - Show how you can edit the parameters easily
2. Show other possible recipes
   - Click "Browse"
   - Show that you can click on them and perform actions on other users' services

Yahoo! Pipes

1. Go to "My Pipes", show the info that they present, and the interactions that you can have with the pipes.

2. Then show that you can get it to output to an RSS feed (or ATOM)

Automator

1. Point out the run buttons at the top, which are the interactions you can have with a completed automation.
2. Show that you access other files in the same way you do with other desktop applications, and point out that this helps with collaboration.

Quartz Composer

1. Again, indicate that they can access files in the same way as a normal desktop application

Questions/Probes

1. How should these services be displayed on screen?
   - Like apps on your phone?
2. What might you want to know about these services?
   - How can you decide whether you want
3. When would you want to set the parameters of the service?
   - As you add that service
   - While you are composing
   - After you have created a composite
4. Other than using the services, would you want to do other things with services you've created?
   - Share it?
   - Clone it?
   - Create an app out of it?
5. What information would you want to record about a service that you've created
   - You might need information to work out what it does when you come back to it in a month's time
   - Maybe you want to share it with one of your friends? What information might they need
6. What interactions might you want to have with those services created by other users?
   - Use them yourself?
   - Copy and customise them?

Get a new service and add it to a composition

Aim for participant: To see how new services can be acquired and used in creating new composite services.

Aim for researcher: To assess the popularity of the suggested service discovery mechanism

Create a composition using the weather service, which you will have to download First, we need you to create a new composite service. Do this in the same way that you did before, and add a component to the composition. This time, tap the "Web" tab in the atomic service list, and select the weather service.

Because the weather service isn't on your device, the tick option isn't available at the top of the atomic service page. Instead, you will see an option to enable you to get the service. Tap this button, and you will be taken to the Google Play App Store entry for the app that contains the weather service. Download this app and then run it, and you should see a message appear at the bottom of the screen to tell you that services have been added on the device.

Now if you go back to the service list, you will see that the weather service has been added into the list of services on the device. Now if you tap the service, you will see that the tick has replace the download button, so you can add the service to the composition

Now you can finish the composition and run it as normal.

Task flow for other applications

Other tools don't have this feature because they don't have a mechanism to acquire new services in the same way our tool does.

Questions / Probes

1. How would you want to acquire services?
    - A separate service store?
    - Through apps
2. Would you be likely to pay for an app if it contained a service that you wanted?
    - I realise this is difficult without a specific example

B Preliminary Requirements List

These requirements were identified in previous studies, either explicitly to gather requirements for EUSC/SC, or those that have specified requirements before performing work on SC.

B.1 Preliminary Functional Requirements

PR1 The application should support each stage of the EUSC life cycle.
All of the stages of the EUSC life cycle should be supported.
Source: [da Silva et al., 2008, Mehandjiev and De Angeli, 2012]

PR1.1 Request / Discovery.
The application should allow users to request and discover components that they can then use in composition to create composites.
Source: [da Silva et al., 2008, Mehandjiev and De Angeli, 2012]

PR1.2 Composition.
The application should allow users to coordinate components to create a composite.
Source: [da Silva et al., 2008, Mehandjiev and De Angeli, 2012]

PR1.3 Verification / Validation.
The application should allow users to verify that the composite they have created executes successfully, and completes the task that they intended.
Source: [da Silva et al., 2008, Mehandjiev and De Angeli, 2012]

PR1.4 Annotation / Deployment.
The application should allow users to add information to the composites they create and then deploy them so that they can be executed.
Source: [da Silva et al., 2008, Mehandjiev and De Angeli, 2012]

PR1.5 Execution.
The application should allow users to execute composites that they've created.
Source: [da Silva et al., 2008, Mehandjiev and De Angeli, 2012]

B.2 Preliminary Non-functional Requirements

PR2 The application should display services by their user interfaces.
Services in the application should be represented by their user interface during the composition process.
Source: [Namoun et al., 2010b, Nestler et al., 2011]

PR3 The composition canvas should be large.
The composition "canvas" should be large enough to allow users to interact with the services on it easily.
Source: [Namoun et al., 2010b]

PR4 Services should be secure.
Services used by the application that require personal information should be secure.
Source: [Namoun et al., 2010b]

PR5 Feedback to users should be continuous and proactive.
Users should be provided with feedback throughout the composition process. This feedback should not have to be prompted.
Source:[Namoun et al., 2010b]

B.3 Preliminary Structural Requirements

PR6 Sets of data should be treated as a single item.
For instance, if a service returns a list of the user's friends, it should only indicate that it returns a single item.
Source: [Mehandjiev et al., 2010b]

PR7 Users should be assisted in resolving problems with dependencies between the invocation of services.
If there are problems with control flow in the composition, e.g. the execution of one service requires the execution of another service that will never be executed, then the user must be assisted with resolving this.
Source: [Mehandjiev et al., 2010b]

PR8 The UI of the composite service being created should be represented.
The user of the application needs to be able to see what the output of the composition process will look like. It should also be linked with the sections in the composition that are presenting that information. Note that this is only applicable to presentation layer composition.
Source: [Mehandjiev et al., 2010b]

PR9 Automate repetitive tasks.
Any tasks in the application that the user has to complete over and over again should be automated where possible.
Source: [Mehandjiev et al., 2010a]

PR10 Users should be able to modify composition templates by removing optional tasks or rearranging the flow of the composition.
If the SC application provides users with templates that they can fill in, they must also be able to edit the order of the components in this template, or remove optional tasks from the template.
Source: [Mehandjiev et al., 2010a, Cappiello et al., 2011b]

PR11 Structured flow options should be provided.
The flow in the structure of the composition should provide structured elements such as looping and branching.
Source: [Albreshne and Pasquier, 2011]

PR12 Allow the configuration of the composition at runtime.
The application should allow the user to configure the composition at runtime as well as at design time.
Source: [Albreshne and Pasquier, 2011]

PR13 Allow multiple instances of the same composition to run at once (with different parameters).
Once the user has created a composite, they should be able to "instantiate" it multiple times with different parameters. For example, they might want to have the same composite that operates across multiple accounts for a social network, so a separate instance would be needed for each account.
Source: [Albreshne and Pasquier, 2011]

PR14 The result of the composition must be usable immediately.
Once the user has created a composite, they must be able to execute it immediately.
Source: [Albinola et al., 2009]

PR15 The outputs of composition should be extremely diverse.
SC applications should allow the user to create a large number of diverse composites.
Source: [Albinola et al., 2009, Aghaee et al., 2012]

PR16 Binding to services should be automatic and switch depending on the availability

of services.
Services that need to be bound (i.e. those in the environment) should do this automatically based on their availability. For instance, if the user moves location then the application should bind with services in that location automatically.
Source: [Bottaro et al., 2007]

PR17 Remote and local services should be represented in the same way.
Services should be represented in the same way regardless of their location.
Source: [Bottaro et al., 2007]

PR18 Abstraction layers should be used to hide complexity.
The inherent complexity of the composition process should be hidden from users by the use of abstraction layers.
Source: [Nestler et al., 2011]

PR19 Code should be hidden from the end-user.
The code that underpins the composition process should be hidden from the user.
Source: [Nestler et al., 2011]

C Requirements List

C.1 Functional Requirements

Specification

R1 Potential compositions could be identified automatically.
The application should be able to monitor the activities of the user and identify tasks that they perform regularly that could be adapted to form a composition.
Rationale: If the application were able to automatically identify potential compositions, it would reduce user burden in deciding what compositions to create.
Category: Functional – Specification
Criticality: Low
Risk: High
Trigger: The user performing a manual task repeatedly.
Preconditions: The application is installed on the user's device.
Postconditions: A composite is created that performs the repetitive task.
Failure effects: The composite is not created.
Associated requirements: None
Source: [Automatic composition identification]

R2 The application should be able to generate a composite from a text description.
Assuming it's possible for the application to generate a description based on the components that are in the composite, this process should be reversible
Rationale: Users may not want to have to manually connect together the components, instead they may just want to describe what they want and have the system create a composite based on this text description.
Category: Functional – Specification
Criticality: Low
Risk: Medium
Trigger: The user entering a description.
Preconditions: The application is installed.
Postconditions: A composite is created.
Failure effects: The composite is not created.
Associated requirements: None *Source:* [Composition to Description ⇒ Description to composition]
Subsequent

Discovery

R3 Users should be able to discover and acquire components.
Users should be able to acquire to acquire new components (either directly or indirectly) through the application.
Rationale: In order to perform composition, users need to first discover the components that they're then able to compose together.
Category: Functional – Discovery
Criticality: Critical
Risk: Low
Trigger: None
Preconditions: application installed

Postconditions: Components discovered
Failure effects: No components discovered; wrong components discovered
Source: [Acquiring components], **R4**, [da Silva et al., 2008, Mehandjiev and De Angeli, 2012, Albreshne and Pasquier, 2011]
Prior identification: [da Silva et al., 2008, Albreshne and Pasquier, 2011, Mehandjiev and De Angeli, 2012]
COTS: AutomateIt, Atooma, IFTTT, Zapier, Yahoo! Pipes, On{X}

R4 Users should be able to search for services.
The application should allow users to search for services within the application.
Rationale: To find either components to be composed, or composites created by others that they can use.
Category: Functional – Discovery
Criticality: High
Risk: Low
Trigger: User enters a search term
Preconditions: application running, connected to component repository, connected to composite repository.
Postconditions: A service is returned.
Failure effects: No service is returned, the wrong service is returned.
Associated requirements: **R4.1**, **R4.1.1**, **R4.1.2**, **R4.2**.
COTS: Quartz Composer, Automator

R4.1 Users should be able to search for components.
The application should allow users to search for services within the application.
Rationale: If the user is looking for a particular component, they need to be able to search for it.
Category: Functional – Discovery
Criticality: High
Risk: Low
Trigger: User entering a search term.
Preconditions: application running, connected to component repository.
Postconditions: A component is returned.
Failure effects: No component is returned, the wrong component is returned.
Associated requirements: **R4.1.1R4.1.2**.
Source: [Search by function]
COTS: Quartz Composer, Automator

R4.1.1 Search for components by function performed.
Users need to be able to search through components as part of the discovery process.
Rationale: If the user has an idea of the function that they want from a component, but not the name of the component, they should be able to search for that function.
Category: Functional – Discovery
Criticality: Low
Risk: Low
Trigger: User enters a search term.
Preconditions: application installed, connected to component repository, component functions documented.

Postconditions: Component retuned.
Failure effects: No component returned, wrong component returned
Associated requirements: None
Source: [Search by name]
COTS: Quartz Composer, Automator

R4.1.2 Search for components by the service provider.
Users should be able to search for components by the application in which the component is provided.
Rationale: If the user knows they want a component provided by a particular service, but does not know its name, they should be able to search for this service provider.
Category: Functional – Discovery
Criticality: Low
Risk: Low
Trigger: User enters a search term.
Preconditions: application installed, connected to component repository, component service providers documented.
Postconditions: Component returned.
Failure effects: No component returned, wrong component returned.
Associated requirements: None
Source: [Search by service provider]

R4.2 Users should be able to search for composites shared by others
Users should be able to search through composites that have been created and shared by other users.
Rationale: Rather than "reinventing the wheel", the user should be able to search for a composite that may have been created by another user to perform the task that they want to perform.
Category: Functional – Discovery
Criticality: Medium
Risk: Low
Trigger: User enters a search term
Preconditions: application installed, connected to composite repository.
Postconditions: Composite returned.
Failure effects: No composite returned, wrong composite returned.
Associated requirements: **R4**.
Source: [Search for shared composites]
COTS: IFTTT, AutomateIt, Yahoo! Pipes, On{X}

R5 Services should be sorted into groups, or ranked.
Lists of services in the application should be separated out into groups, or ranked based on some property of the service.
Rationale: Users should be able to browse through components if they don't know exactly what they are looking for, and a grouping or ranking mechanism would make the browsing process easier.
Category: Functional – Discovery
Criticality: High
Risk: Low
Source: [Grouping]
Prior identification: [Bottaro et al., 2007] *COTS:* Tasker, Atooma, IFTTT, Zapier, Yahoo! Pipes, Automator, On{X}

Associated requirements: **R5**, **R5.1**-**R5.8**.

R5.1 Services should be grouped by their function.
Services should be grouped based on the function that they perform. For example, all of the components involving interactions with social media could be in a social category.
Rationale: Services can be grouped by their function in order to collect similarly-functioning services together.
Category: Functional – Discovery
Criticality: Medium
Risk: Low
Source: [Grouping by function]
COTS: Tasker, Automator, Yahoo! Pipes, On{X}
Associated requirements: **R5**, **R5.2**-**R5.8**.

R5.2 Components should be grouped by their location.
Components should be put into groups based on where they can be found. For example, components on the device might be in one group, and components from the web in another.
Rationale: The user may want to distinguish between components in a local location over those in a remote location.
Category: Functional – Discovery
Criticality: Low
Risk: Low
Source: [Grouping by component location]
Associated requirements: **R5**, **R5.1**, **R5.3**,**R5.8**.

R5.3 Components should be grouped by the app that provides them.
Within their location, components should be grouped by the application that provides them.
Rationale: the user may want to distinguish between components provided by apps that they already have over those that they don't.
Category: Functional – Discovery
Criticality: Low
Risk: Low
Source: [Grouping by the containing application]
Dependency: Composer component distribution paradigm
COTS: Automator
Associated requirements: **R5**, **R5.1**, **R5.2**, **R5.4**-**R5.8**.

R5.4 Components should be grouped by their cost.
Components should be grouped by their cost, or the cost of the application that provides the component.
Rationale: Users may want to distinguish between components that will cost them money and those that won't.
Category: Functional – Discovery
Criticality: Low
Risk: Low
Source: [Grouping by cost]
Associated requirements: **R5**, **R5.1**-**R5.3**, **R5.5**-**R5.8**.

R5.5 Components grouped by their rating.
Components should be grouped by their rating – top rated services should be shown in a separate section.

Rationale: Users may want to distinguish between components whose quality has been assessed as being higher or lower than others.
Category: Functional – Discovery
Criticality: Medium
Risk: Low
Source: [Top Rated]
Associated requirements: **R5**, **R5.1**-**R5.4**, **R5.6**-**R5.8**.

R5.6 Components should be grouped by the service provider.
Components should be grouped by the provider of the service, or the developer of the service.
Rationale: If browsing for a component provided by a particular service, the user may want to find other components that are also provided by that service.
Category: Functional – Discovery
Criticality: Medium
Risk: Low
Source: [Grouping by service provider]
COTS: Atooma, IFTTT, Zapier
Associated requirements: **R5**, **R5.1**-**R5.5**, **R5.7**, **R5.8**.

R5.7 Components should be ranked by previously used.
Components should be grouped to show those that have been used previously by the user.
Rationale: If a user has previously used components, they may want to find these components again with little effort.
Category: Functional – Discovery
Criticality: Medium
Risk: Low
Source: [Previously used components, Recently used]
Associated requirements: **R5**, **R5.1**-**R5.6**, **R5.8**.

R5.8 The application shouldn't allow users to group services.
Services shouldn't be grouped.
Rationale: Users may not want to have any sort of grouping imposed on their browsing of services.
Category: Functional – Discovery
Criticality: Medium
Risk: Low
Source: [No grouping]
COTS: Quartz Composer
Associated requirements: **R5**, **R5.1**-**R5.7**.

R6 Services should be grouped into groups and sub-groups.
Grouping should allow for services to be put into groups and then sub-groups.
Rationale: Grouping metrics may have more than one level. Grouping should reflect this.
Category: Functional – Discovery
Criticality: Low
Risk: Low
Source: [Groups and sub-groups]
Associated requirements: **R5**, **R5.1**-**R5.8**.

R7 Services should be "tagged" with keywords to represent their function.
Instead of being grouped, services should be assigned tags describing their functional-

ity.
Rationale: Grouping can be restrictive, tagging is more flexible.
Category: Functional – Discovery
Criticality: Low
Risk: Low
Source: [Tagging]
COTS: Atooma
Associated requirements: None

Composition – Design & Construction

R8 The application must allow users to compose services.
Users must be able to create composite services from component services using the application – the primary function of an EUSC application.
Rationale: This is the main function of a EUSC application
Category: Functional – Composition
Criticality: Critical
Risk: Low
Trigger: The user chooses to create a composite.
Preconditions: The application is installed.
Postconditions: A composite is created.
Failure effects: A composite is not created.
Associated requirements: **R9**
Source: [Composition]
COTS: Atooma, Tasker, IFTTT, Zapier, Yahoo! Pipes, Quartz Composer, Automator, On{X}

R9 Users should be able to edit the order of the components in composition.
If the components in the composition are positioned in a particular way, the user should be able to modify this initial order.
Rationale: The user may position the components in the wrong order initially, or change their mind about what order they want the components to execute in.
Category: Functional – Composition
Criticality: High
Risk: Low
Trigger: The user moves a component.
Preconditions: There are components in the composition.
Postconditions: The components in the composition are in the new order.
Failure effects: The components in the composition are in their original order.
Associated requirements: **R8**.
Source: [Editing composition, Customisation]
COTS: Tasker, Yahoo! Pipes, Quartz Composer, Automator

R10 Composition must not involve coding.
The composition process should not involve the user having to write any code
Rationale: EUSC is designed for users who are not skilled enough in programming to create the composite manually.
Category: Functional – Composition
Criticality: Critical
Risk: Low
Trigger: The user indicates they want to create a composite.

Preconditions: The application is installed.
Postconditions: A composite is created.
Failure effects: A composite is not created.
Associated requirements: None
Source: [Composition – no coding]
Prior identification: [Namoun et al., 2010b, Nestler et al., 2011, Picozzi, 2010, Albreshne and Pasquier, 2011, Cappiello et al., 2011a]
COTS: Atooma, Tasker, IFTTT, AutomateIt, Zapier, Yahoo! Pipes, Quartz Composer, Automator

R11 The application should facilitate complex compositions to give the user more power.
The application should allow the user to create complex compositions so that they can create a wider range of more powerful composites.
Rationale: EUSC seeks to allow end users to create applications/services that they would otherwise have to ask a developer to make for them. SC should allow such complexity in the composition.
Category: Functional – Composition
Criticality: Medium
Risk: Low
Trigger: The user indicates they want to create a composite.
Preconditions: The application is installed.
Postconditions: A composite is created.
Failure effects: A composite is not created.
Associated requirements: None
Source: [Complexity is power]

R12 The composition process should be semi-automated.
The process of composition should be semi-automatic to assist the user during composition.
Rationale: Users may need assistance with matching of inputs and outputs in the composition process.
Category: Functional – Composition
Criticality: High
Risk: Low
Prior identification: [Mehandjiev et al., 2010b, Namoun et al., 2010b]
COTS: Atooma, AutomateIt, IFTTT, Zapier, Yahoo! Pipes, Quartz Composer

R13 Composition should work with components whose data types do not match.
Composition of components that have data types that do not match should be allowed since in some cases the execution of a subsequent service does not rely on the data being passed to it by a preceding service.
Rationale: There may be instances where the data is not being used, so the type of that data does not matter.
Category: Functional – Composition
Criticality: High
Risk: Low
Source: [Using components that don't match]
Prior identification: [Nestler et al., 2011]
COTS: Automator, On{X}
Associated requirements: **R31.1**

Verification & Validation

R14 The application should allow users to test composites.
The application should allow users to test their composites-in-progress whilst they are creating them.
Rationale: Users need to ensure that composites they create function as intended.
Category: Functional – Verification
Criticality: Critical
Risk: Low
Trigger: User indicates that they want to test the composite.
Preconditions: The composite contains some components.
Postconditions: The composite executes in its current state.
Failure effects: The component doesn't execute as intended
Associated requirements: **R15**-**R17**
Source: [Testing]
COTS: Yahoo! Pipes, Quartz Composer, Automator, Tasker, Yahoo! Pipes, On{X}

R15 Components should have a test mode.
A "test mode" should be provided with components that interact with external entities.
Rationale: Some components might have an action that operates on an external entity, but while being tested, the user might not want the component to have any interaction with that external entity. For instance, sending a SMS.
Category: Functional – Verification
Criticality: Medium
Risk: Low
Trigger: User indicates that they want to test the composite.
Preconditions: The composite contains some components.
Postconditions: The composite executes in its current state.
Failure effects: The component doesn't execute as intended
Associated requirements: **R15.1**-**R15.3**, **R16**, **R17**
Source: [Test mode for components]

R15.1 Testing of triggers should be simulated.
Compositions that are created with triggers to initialise them would only normally be executed when that trigger is fired. Even though this can be done manually in some cases, for testing purposes, users should be able to simulate that this trigger has occurred.
Rationale: Triggers (components that execute when a particular event occurs) need the event to occur before they can be executed – restricting when they can execute. This allows them to be tested.
Category: Functional – Verification
Criticality: High
Risk: Low
Trigger: User indicates that they want to test the composite.
Preconditions: The composite contains some components, the first of which is a trigger.
Postconditions: The component executes in its current state.
Failure effects: the component doesn't execute.
Associated requirements: **R15**
Source: [Simulate testing of triggers]

R15.2 Dummy data should be provided for testing.

Some services can acquire data from an external data source; when being tested, these services should provide "dummy" data.
Rationale: Components may need to look up data in order to pass it to other components – the user should be able to verify that this will work without having to commit to looking up the data.
Category: Functional – Verification
Criticality: Medium
Risk: Low
Trigger: User indicates that they want to test the composite.
Preconditions: The composite contains some components.
Postconditions: The component executes in its current state.
Failure effects: the component doesn't execute.
Associated requirements: **R15**
Source: [Dummy data for testing]

R15.3 Components should indicate their execution process while they are being tested.
Whilst a component is being tested, it should report the task that is being performed to the composition application so that the user can better determine where errors occur.
Rationale: The user may need more information regarding what is happening within a component – for instance if it fails they would be able to see where it failed.
Category: Functional – Verification
Criticality: Medium
Risk: Low
Trigger: User indicates that they want to test the composite.
Preconditions: The composite contains some components.
Postconditions: The component executes in its current state.
Failure effects: the component doesn't execute.
Associated requirements: **R15**
Source: [Execution progress within component]
COTS: Automator
Subsequent

R16 Composition should be debuggable.
The application should allow the user to debug the composition while they are creating it.
Rationale: If testing the composition fails, the user should be able to step through the execution to find out where it failed.
Category: Functional – Verification
Criticality: Medium
Risk: Low
Trigger: User indicates that they want to debug the composite.
Preconditions: composite contains some components, the first of which is a trigger.
Postconditions: The component executes in its current state.
Failure effects: the component doesn't execute.
Associated requirements: **R15**
Source: [Debugging]
COTS: Yahoo! Pipes

R17 Testing should only be needed when the composition is sufficiently complicated.

Testing is only needed in a composition application where the composites being created are sufficiently complex.
Rationale: If a composite is very straight forward, it should not need to be tested.
Category: Functional – Verification
Criticality: Low
Risk: Low
Associated requirements: **R15**
Source: [Testing complexity]
COTS: Yahoo! Pipes, IFTTT

Annotation & Deployment

R18 The application should integrate with users' social networks.
Users should be able to log in to the application through one or more social networks and connect with the friends that they have on these networks.
Rationale: Users may want to share their creations with their friends (or discover friends' creations).
Category: Functional – Annotation & deployment
Criticality: Medium
Risk: Medium
Trigger: The user enters their social network details.
Preconditions: The application is installed, the user is registered on the social network.
Postconditions: The user is connected to the social network through the application.
Failure effects: The user and application are not connected to the social network
Associated requirements: **R23**
Source: [View friends' composites, Sharing/publishing of composites on Social Networks]
COTS: Atooma

R19 Descriptions of composites should be generated automatically.
The application should be able to generate descriptions for composites based on the descriptions of the components that make them up and the logical operations that combine them.
Rationale: The user may not want to think of descriptions for the composites they create, so the application should provide a facility to do this for them.
Category: Functional – Annotation & Deployment
Criticality: Low
Risk: Medium
Trigger: The user saves their composite.
Preconditions: The user has created a composite containing various components.
Postconditions: The composite then gets an automatically generated description.
Failure effects: The composite does not get a description.
Associated requirements: **R45.2**
Source: [Automatically generate descriptions for composites]

Monitoring & Adaptation

R20 The application should allow users to delete composites.
The application should allow users to delete composites that they have acquired or created.

Rationale: Once a composite has been used and is no longer required, the user should be able to delete it.
Category: Functional – Management – Monitoring & Adaptation
Criticality: High
Risk: Low
Trigger: The user selects delete.
Preconditions: There is a composite to delete.
Postconditions: The composite is deleted.
Failure effects: The composite is not deleted.
Associated requirements: None
Source: [deleting entities]
COTS: Atooma, Tasker, AutomateIt, IFTTT, Zapier, Yahoo! Pipes, Automator, On{X}

R21 The application should allow users to execute composites.
The application must be able to execute composites that they have acquired or created.
Rationale: Composites need to be executed.
Category: Functional – Management – Monitoring & Adaptation
Criticality: Critical
Risk: Low
Trigger: The user chooses to run the composite, or the trigger in the composite is activated by an external event.
Preconditions: A composite has been created.
Postconditions: The composite is executed.
Failure effects: The composite is not executed.
Associated requirements: None
Source: [Testing]
COTS: Atooma, Tasker, AutomateIt, IFTTT, Zapier, Yahoo! Pipes, Quartz Composer, Automator, On{X}

R22 The application should allow users to share services.
The application should allow users to share services that are created using the application with other users of the application.
Rationale: Once a user has created and used a composite, they might want to share it with others.
Category: Functional – Management – Monitoring & Adaptation
Criticality: High
Risk: Low
Trigger: The user indicates they want to share the composite.
Preconditions: There is a composite to share.
Postconditions: The composite is shared to a shared composite repository.
Failure effects: The composite isn't shared.
Associated requirements: **R18**, **R23**
Source: [Sharing/publishing of composites]
COTS: Atooma, IFTTT, Yahoo! Pipes, AutomateIt, On{X}

R23 The application should allow users to share composites on social networks.
Users should be able to share what they create on social networks so their friends can interact with them.
Rationale: Users may want to show off their creations with their friends on social networks.
Category: Functional – Management – Monitoring & Adaptation
Criticality: Medium

Risk: Low
Trigger: The user indicates that they want to share the composite.
Preconditions: There a composite to share, the application is connected to the social network.
Postconditions: The composite is shared to the user's social network.
Failure effects: The composite is not shared.
Associated requirements: **R18**, **R22**
Source: [Sharing/publishing of composites on Social Networks]
COTS: IFTTT
Subsequent

R24 The application should allow users to rate services.
Users should be able to rate services to convey their opinion on the quality of the service to other users of the application.
Rationale: Users should be able to provide feedback to the creators of components and composites.
Category: Functional – Management – Monitoring & Adaptation
Criticality: Medium
Risk: Low
Trigger: The user indicates they want to rate the service.
Preconditions: There is a service to be rated.
Postconditions: The service's current rating gets aggregated with the new rating.
Failure effects: The rating is not applied.
Associated requirements: **R25**
Source: [User ratings]
COTS: AutomateIt, IFTTT, Yahoo! Pipes, On{X}

R25 The application should allow users to interact with composites created by other users.
Users should be able to interact with the services that other users create and share.
Rationale: Users shouldn't need to "reinvent the wheel" and also be able to provide feedback on these composites.
Category: Functional – Management – Monitoring & Adaptation
Criticality: Medium
Risk: Low
Trigger: The user indicates that they want to browse through created composites.
Preconditions: There are composites available to be browsed through.
Postconditions: None
Failure effects: None
Associated requirements: **R22**
COTS: IFTTT, Yahoo! Pipes, Atooma, AutomateIt, Zapier, Yahoo! Pipes
Source: [Sharing]

R25.1 Users should be able to interact with composites created by their friends.
Users should be able to view the composites that their friends have created, and interact with them in the same way they can interact with others.
Rationale: Users may want to interact with composites created by people they know, rather than all users of the application.
Category: Functional – Management – Monitoring & Adaptation
Criticality: Low
Risk: Low
Trigger: The user indicates that they want to browse through created composites.

Preconditions: There are composites available to be browsed through.
Postconditions: None
Failure effects: None
Associated requirements: **R22**
Source: [View friends' composites]
Dependent: **R25**

R26 Services should be customisable by editing their parameters.
Users should be able to customise services by editing their parameters.
Rationale: Components and composites may need to be customised after they have been created.
Category: Functional – Management – Monitoring & Adaptation
Criticality: Medium-High
Risk: Low
Trigger: The user chooses to set the parameters of the service.
Preconditions: There is a service to set the parameters of.
Postconditions: The parameters of the service are set.
Failure effects: The parameters are not set.
Associated requirements: **R26.1.2**-**R26.3.2**
Source: [Parameters, Component customisation], **R26.1.2**-**R26.3.2**
COTS: Yahoo! Pipes, Quartz Composer, Tasker, IFTTT, Atooma, Automator, AutomateIt, Zapier, On{X}

R26.1.2 Parameters should be editable during and after composition time, and up to and including at runtime.
Users should be able to edit the parameters of a composite after the initial composition process.
Rationale: The user might want to change the parameters at any time.
Category: Functional – Management – Monitoring & Adaptation
Criticality: Medium
Risk: Low
Trigger: The user changes the parameters.
Preconditions: The composite is in an executable form.
Postconditions: The composite has its parameters set.
Failure effects: The parameters are not set.
Associated requirements: **R26**, **R26.2.2**, **R26.3.2**
Source: [Set parameters after composition, Set parameters at runtime, Set parameters at composition time]
COTS: Yahoo! Pipes, Quartz Composer, Tasker, IFTTT, Atooma, Automator, AutomateIt, On{X}

R26.2.2 Parameters should be set to default values at composition time and be editable later.
Users should be able to set default values for parameters at composition time, which can then be edited later.
Rationale: The user should be able to run the component without having to explicitly set the parameters.
Category: Functional – Management – Monitoring & Adaptation
Criticality: Medium
Risk: Low
Trigger: The user runs the composite.
Preconditions: The composite has been created.

Postconditions: The composite executes with default parameters.
Failure effects: None
Associated requirements: **R26**, **R26.1.2**, **R26.3.2**
Source: [Set default parameters at composition-time, Parameters – don't ask again]
COTS: Tasker, IFTTT, Atooma, AutomateIt, Zapier, Quartz Composer, Automator, On{X}

R26.3.2 Setting of parameters should use the composite description.
Setting the parameters of the composite should involve changing the description of the composite.
Rationale: Using the description to modify the parameters is intuitive, and the description is updated at the same time.
Category: Functional – Management – Monitoring & Adaptation
Criticality: Low
Risk: Low-Medium
Trigger: The user modifies the description.
Preconditions: The composite has been created.
Postconditions: The description is changed, the parameters are changed.
Failure effects: The description and parameters are not changed.
Associated requirements: **R26**, **R26.1.2**, **R26.2.2**
Source: [Parameter setting in On{X}]
COTS: On{X}

R27 Components should only need to be activated once.
If the user needs to activate the component i.e., entering usernames and passwords, this should only need to be done once no matter how many composites into which the component is composed.
Rationale: It would be annoying for the user to have to activate components every time they had to use them rather than just once.
Category: Functional – Management – Monitoring & Adaptation
Criticality: High
Risk: Low
Trigger: The user activates the component.
Preconditions: The component needs to be activated.
Postconditions: The component is activated.
Failure effects: The component is not activated.
Associated requirements: None
Source: [Component activation]
COTS: Atooma, Tasker, AutomateIt, IFTTT, Zapier, Yahoo! Pipes, Quartz Composer, Automator

C.2 Non-Functional Requirements

Interaction Requirements

R28 The process of composition should be drag-n-drop editable.
The user should be able to re-order and re-position components in the composition by dragging-and-dropping the components.
Rationale: Moving components around a canvas was deemed to be a simple and effective way of interacting with the component.

Category: Non-Functional – Interaction
Criticality: Medium
Risk: Low
Associated requirements: **R9**
COTS: Yahoo! Pipes, Quartz Composer
Source: [Drag-and-drop editing]
Subsequent

Operability

Helpfulness

R29 The application should be linked to an associated forum.
The application should provide a forum to allow users to discuss problems that they are having with the application with other users and experts.
Rationale: Users can discuss problems, ideas, etc. on a forum with other users of the application.
Category: Non-Functional – Operability - Helpfulness
Criticality: Medium
Risk: Low
Associated requirements: None
Source: [Forum]
COTS: On{X}, Zapier, Yahoo! Pipes
Subsequent

R30 The application should be linked to an associated blog.
The application should provide a blog to keep users up-to-date with the application and any new features to be implemented.
Rationale: Users should be able to keep up to date with changes in the application through a blog.
Category: Non-Functional – Operability - Helpfulness
Criticality: Medium
Risk: Low
Associated requirements: None
Source: [Blog]
COTS: On{X}, IFTTT, Zapier
Subsequent

R31 Assistance should be provided during composition.
The application should provide the user with assistance while they are performing composition, for instance automation or interactive help applications.
Rationale: Composition is inherently difficult, so the user may require assistance at various points throughout the process.
Category: Non-Functional – Operability - Helpfulness
Criticality: High
Risk: Low
Associated requirements: **R31.1**, **R31.1.1**-**R31.1.3**
Source: [Assistance provided]
COTS: IFTTT, Atooma, AutomateIt, Zapier, Yahoo! Pipes
Subsequent

R31.1 Assistance should be provided with inputs and outputs.

The application should assist the user with inputs and outputs, with mechanisms such as data-type matching.
Rationale: Inputs and outputs are one of the more technical aspects of SC, and participants indicated that this is an element that they would require assistance with.
Category: Non-Functional - Operability – Helpfulness
Criticality: High
Risk: Low
Associated requirements: **R31.1.1**, **R31.1.2**, **R31.1.3**
Source: [Assistance with input/output]
Prior identification: [Albreshne and Pasquier, 2011]
COTS: Yahoo! Pipes, Quartz Composer

R31.1.1 Components that don't match should be shown.
The application should present components to the user where the data type of the input doesn't match with the output of the component in the previous position so that they can use them in the composite regardless.
Rationale: Components may not have data types that match, but the user still wants to use them in a composite.
Category: Non-Functional – Operability - Helpfulness
Criticality: High
Risk: Low
Associated requirements: **R31.1**, **R31.1.2**, **R31.1.3**
Source: [Show/hide components that don't match, Using components that don't match]
COTS: Yahoo! Pipes, Quartz Composer

R31.1.2 Components that won't work at all should be hidden.
The application should hide components that don't work in the selected position. For example, triggers only work in the first position of the composition.
Rationale: Some components will not work at all in a particular position (e.g. triggers only work in the first position).
Category: Non-Functional – Operability - Helpfulness
Criticality: High
Risk: Low
Associated requirements: **R31.1.1**, **R31.1.3**
Source: [Show/hide components that don't work]
COTS: Atooma, IFTTT, Zapier

R31.1.3 Components whose inputs match outputs in the composition should be presented to users.
Users should be presented with services whose inputs and outputs match with services currently in the composition to give them inspiration.
Rationale: The users should be given some assistance as to which components they may want to choose to add to the composition.
Category: Non-Functional – Operability - Helpfulness
Criticality: Medium
Risk: Low
Associated requirements: **R31.1.1**, **R31.1.2**
Source: [Matching services – inspiration]
Prior identification: [2]

COTS: Atooma, IFTTT, Zapier, Quartz Composer

R32 The application should have a tutorial or instruction page.
The application should provide a tutorial section or instruction page so that users who have never used the application before can learn how to use it.
Rationale: Inexperienced users may need instructions as to how to use the application, or tutorials to walk them through how to use it.
Category: Non-Functional – Operability - Helpfulness
Criticality: Medium
Risk: Low
Associated requirements: None
Source: [Tutorial/instruction page]
COTS: IFTTT, Atooma, Tasker, AutomateIt, Yahoo! Pipes, Quartz Composer, Automator, On{X}

R33 Warnings should be used to illustrate potential problems in composition.
The application should display warnings to the user when potential problems are identified in the composition. For example, when services are connected together and the data types do not match.
Rationale: Users need to know when something might not work as they expect, and a warning can inform them of this.
Category: Non-Functional - Operability – Helpfulness
Criticality: High
Risk: Low
Associated requirements: None
Source: [Warnings]
Prior identification: [[Mehandjiev et al., 2010a]
COTS: Atooma, AutomateIt
Subsequent

R34 Templates should be provided.
The application should provide templates to users that they can slot components into to simplify the composition process.
Rationale: Templates can make the process of composition easier, by asking the user to slot components into the composition rather than having to position the components themselves.
Category: Non-Functional – Operability - Helpfulness
Criticality: Medium-High
Risk: Low
Associated requirements: **R34.1**
Source: [Templates]
Prior identification: [[Mehandjiev et al., 2010b]
COTS: Atooma, IFTTT, Automator, Tasker, Zapier, AutomateIt

R34.1 Linear templates should be provided.
For simplicity, one of the templates provided by the application should be linear, meaning no branching or looping is supported.
Rationale: Linearity is simple and easily followed by our participants.
Category: Non-Functional – Operability - Helpfulness
Criticality: Medium
Risk: Low
Associated requirements: **R34**
Source: [Linearity of composition]

COTS: Atooma, IFTTT, Automator, Tasker, Zapier, AutomateIt

R35 There should be an option for composition to be unrestricted.
The application should give the user complete control over the composition without any restrictions so that they have more control over the output of the process.
Rationale: As well as restricting composition with templates, advanced users should be able to perform composition in a free-form manner.
Category: Non-Functional – Operability - Helpfulness
Criticality: Medium
Risk: Low
Associated requirements: None
Source: [Free-form composition]
COTS: Yahoo! Pipes, Quartz Composer
Conflict: **R52**

R36 Recommendations/Examples of potential compositions should be presented to users.
Users should be presented with examples of potential components and/or compositions to give them inspiration, for clarity/instruction, and to ensure that they don't "re-invent the wheel".
Rationale: Recommendations and examples give users help with what components they might want to add to the composition, or an idea of what they might want to create overall.
Category: Non-Functional – Operability - Helpfulness
Criticality: High
Risk: Low
Associated requirements: **R36.1**, **R36.2**
Source: [Examples, Examples - don't re-invent the wheel, Examples for inspiration, Examples for clarity/instruction, Recommendations from the Internet]
COTS: Atooma, IFTTT, Zapier, Quartz Composer, Automator, On{X}

R36.1 Users should be able to get recommendations from friends.
Users should be able to see composites and components that their friends have recommended.
Rationale: Users are more likely to trust their friends, so are likely to follow recommendations based on what their friends have made.
Category: Non-Functional – Operability - Helpfulness
Criticality: Low
Risk: Low
Associated requirements: **R36**, **R36.2**
Source: [Recommendations from friends]
COTS: On{X}
Subsequent

R36.2 Examples should be presented first.
Users should be presented with examples as the first thing they see in the composition application since they can perform many functions to aid the user.
Rationale: Examples of potential composites are a good place to start as they give the user inspiration as to what they might want to create.
Category: Non-Functional – Operability - Helpfulness
Criticality: High
Risk: Low

Associated requirements: **R36**, **R36.1**
Source: [Examples first]
COTS: IFTTT, Quartz Composer

Technical Accessibility

R37 Terminology should be simple, user-friendly and consistent.
The terminology used in the simple needs to be simple in order to minimise user confusion.
Rationale: Non-technical users need to be able to understand how to use the application, and user friendly terminology can help to explain it.
Category: Non-Functional – Operability – Technical accessibility
Criticality: Critical
Risk: Low
Associated requirements: None
Source: [Terminology, Terminology confusion, Terminology consistency, Input/output vs. parameter, Parameters are like settings]
Prior identification: [Namoun et al., 2010b]
COTS: IFTTT

R38 Two application views or versions: high-tech and low-tech.
Users with varying levels of technical ability are unlikely to be supported in the right way by the same application or view of application. Then different users could be provided with different properties and levels of technical information.
Rationale: Given the differences in technical ability between potential users, and the difference in support that they require, it may be easier to create two different versions of the application (or views on the application) to separate support for each of these types of user.
Category: Non-Functional – Operability – Technical accessibility
Criticality: Low
Risk: Low
Associated requirements: None
Source: [Two versions: high tech and low tech, Tasker variables, variable comparison in Tasker]
Prior identification: [Albinola et al., 2009, Aghaee et al., 2012]

Appropriateness Recognisability

R39 Colour should be used within the composition process.
Colour should be used to distinguish between different aspects of the composition, for instance types of input or output.
Rationale: Colour is a mechanism that can be used to identify different aspects of the composition – such as matching data types.
Category: Non-Functional – Operability- Appropriateness recognisability
Criticality: Medium
Risk: Low
Associated requirements: None
Source: [Use of colour]
COTS: Atooma, AutomateIt

R40 The composition process should be very visual.
The representation of the composition process should be visual to encourage users to interact with it.
Rationale: A more visual experience may make the process more appealing.
Category: Non-functional – Operability – Appropriateness recognisability
Criticality: Low
Risk: Low
Associated requirements: None
Source: [Visual]
Prior identification: [Nestler et al., 2011]
COTS: Atooma, IFTTT, Yahoo! Pipes, Quartz Composer

R41 The representation of components and composites should be different.
The application should assist users in distinguishing between components and composites by ensuring that their representation is different.
Rationale: The user needs to be able to easily distinguish between components and composites.
Category: Non-functional – Operability – Appropriateness recognisability
Criticality: High
Risk: Low
Associated requirements: None
Source: [Different view for composites and components]
COTS: Atooma, Tasker, AutomateIt, IFTTT, Zapier, Yahoo! Pipes

R42 Inputs and outputs of components should be presented to users.
The application should present the inputs and outputs of components to users, to aid the process of connecting components together (if data passing is supported). They can also be used to help users determine the function of the component.
Rationale: If the components can pass data between one another, this should be presented to users so that they can understand what data is being passed between them.
Category: Non-Functional - Operability – Appropriateness recognisability
Criticality: High
Risk: Low
Associated requirements: **R42**.*
Source: [Input/output, Input/output as function]
COTS: IFTTT, Yahoo! Pipes, Quartz Composer, Automator, On{X}

R42.1 Inputs and outputs should convey the type of data required.
Inputs and outputs of components should present their data-type to assist the user with performing data-type matching and to reduce ambiguity in composition.
Rationale: If the user is to deal with the inputs and outputs of components, they need to be aware of the data that is being passed between them.
Category: Non-Functional - Operability – Appropriateness recognisability
Criticality: High
Risk: Low
Associated requirements: **R42.2**-**R42.4**
Source: [Data types, matching data types]
COTS: Quartz Composer, IFTTT, Yahoo! Pipes, Automator, On{X}

R42.2 The application should distinguish between mandatory and optional inputs.
Components may have inputs that are required for the service to operate success-

fully, whereas some inputs might be optional. The application should indicate this difference to the user.
Rationale: The user needs to know which inputs and outputs they must deal with vs. those that they don't have to set.
Category: Non-Functional – Operability – Appropriateness recognisability
Criticality: High
Risk: Low
Associated requirements: **R42.1**, **R42.3**, **R42.4** *Source:* [Mandatory vs. optional inputs]
COTS: IFTTT, Quartz Composer, Automator

R42.3 Matching data types should be indicated.
The application should indicate when there is a match of the data-types of two or more inputs or outputs.
Rationale: If the user is expected to connect inputs and outputs together, they need to be able to tell which will connect with one another.
Category: Non-Functional – Operability – Appropriateness recognisability
Criticality: High
Risk: Low
Associated requirements: None
Source: [Assistance with input/output]
Prior identification: **R42.1**, **R42.2**, **R42.4**
COTS: Yahoo! Pipes, Quartz Composer

R42.4 Inputs and outputs might only need to be presented if there is a problem in the coordination.
Inputs and outputs should only be presented to users if there is an issue with ambiguity or data matching.
Rationale: Inputs and outputs could be connected together automatically where possible, but would need to be confirmed by users if there is no match or ambiguity.
Category: Non-Functional – Operability – Appropriateness recognisability
Criticality: Low
Risk: Low
Associated requirements: **R42.1**-**R42.3**
Source: [Doesn't care if input/output match, Ambiguity in inputs/outputs]

R43 Connections between components should be represented explicitly to the user.
The connections that are created between services in the composition process should be explicitly represented to the user.
Rationale: Users need to be able to determine which components have connections between them and which do not.
Category: Non-functional – Operability – Appropriateness recognisability
Criticality: Medium
Risk: Low
Associated requirements: **R43**.*
Source: [Explicit connections between components]
COTS: AutomateIt, Atooma, IFTTT, Zapier, Yahoo! Pipes, Quartz Composer, Automator

R43.1 Successful connections should be represented.
The application should indicate to the user if a successful connection is made between two components in the composition as opposed to an unsuccessful

connection.
Rationale: Users need to be able to tell if connections have been made successfully.
Category: Non-functional – Operability – Appropriateness recognisability
Criticality: Medium
Risk: Low
Associated requirements: **R43**
Source: [Indicator of successful connection between components]
COTS: Quartz Composer, Yahoo! Pipes

R43.2 Connections should be implicit.
The connections that are created between services in the composition process should not be represented explicitly to the user, and should instead be implicit when connections between components are present.
Rationale: Connections between components aren't important unless there is a problem.
Category: Non-functional – Operability – Appropriateness recognisability
Criticality: Low
Risk: Low
Associated requirements: **R43**
Source: [Implicit connections between components]
COTS: Tasker, Atooma, On{X}

R44 Flow should be represented in composition.
The application should represent flow between components to the user.
Rationale: Users needs to be able to be able to determine either what order the components will execute in, or the data that is passed between them.
Category: Non-Functional – Operability – Appropriateness recognisability
Criticality: Medium
Risk: Low
Associated requirements: **R44.1**-**R44.3**
Source: **R44.1**-**R44.3**, [Flow diagram]
COTS: IFTTT, Tasker, Atooma, AutomateIt, Automator, Zapier, Yahoo! Pipes, Quartz Composer

R44.1 The flow of control between components should be represented in composition. The application should present the order that the components execute in.
Rationale: Users need to be able to identify the order in which components are executed in the composition
Category: Non-Functional – Operability – Appropriateness recognisability
Criticality: High
Risk: Low
Associated requirements: **R44**, **R44.2**, **R44.3**
Source: [Control flow]
Prior identification: [Mehandjiev et al., 2010b]
COTS: Atooma, Tasker, AutomateIt, IFTTT, Zapier, Automator

R44.2 The flow of data between components should be represented in composition, if it is present.
The application should show how the data is passed between the components in the composition.
Rationale: Users should be able to identify what data is being passed between components in the composition

Category: Non-Functional – Operability – Appropriateness recognisability
Criticality: Medium
Risk: Low
Associated requirements: **R44**, **R44.1**, **R44.3**
Source: [Data flow]
Prior identification: [Mehandjiev et al., 2010b]
COTS: Yahoo! Pipes, Quartz Composer

R44.3 Flow representation should be unambiguous.
The representation of flow should be unambiguous.
Rationale: Whichever type of flow is represented or present, its representation needs to be one that can be understood by users.
Category: Non-Functional – Operability – Appropriateness recognisability
Criticality: High
Risk: Low
Associated requirements: None
Source: [Ambiguity of flow]
COTS: Atooma, Tasker, AutomateIt, IFTTT, Zapier

R45 The application should present attributes of services that can help them determine its function.
All services in the application have attributes that the user may need to be aware of.
Rationale: The application should present attributes of services so that users can determine whether (a) they want to use the components in composition, or (b) whether they want to acquire a composite that another user has made.
Category: Non-Functional – Operability – Appropriateness recognisability
Criticality: Critical
Risk: Low
Associated requirements: None
Source: **R45**.*

R45.1 The application should present the names of services.
Services in the application should present their name to the user.
Rationale: The user needs to know what the service is called so they can identify it.
Category: Non-Functional – Operability – Appropriateness recognisability
Criticality: Critical
Risk: Low
Associated requirements: None
Source: [Name]
COTS: Atooma, AutomateIt, IFTTT, Yahoo! Pipes, Quartz Composer, Automator, Tasker

R45.2 The application should present descriptions of services.
Services in the application should provide a description of what they do.
Rationale: The user needs to be able to determine what the service does, and if they are not able to learn this from the name of the service alone, a description of what the service does would provide this.
Category: Non-Functional – Operability – Appropriateness recognisability
Criticality: Critical
Risk: Low
Associated requirements: **R45.2.1**
Source: [Description, Short description]

COTS: IFTTT, Quartz Composer, Automator

R45.2.1 The application should present short descriptions of services.
As well as providing a full description, the application should allow services to provide a short one-line description.
Rationale: The user needs to be able to quickly get an idea of what the service does, so descriptions should not be verbose.
Category: Non-Functional – Operability – Appropriateness recognisability
Criticality: Medium
Risk: Low
Associated requirements: **R45.2**
Source: [Description, Short description]

R45.3 The application should present an icon for the service.
Services in the application should present an icon as a graphical representation.
Rationale: Users are likely to associate images with different services that can be provided.
Category: Non-Functional – Operability – Appropriateness recognisability
Criticality: High
Risk: Low
Associated requirements: None
Source: [Icon]
COTS: Atooma, IFTTT, AutomateIt, Automator

R45.4 The application should present screenshots of services.
Services in the application should show screenshots of the component being used, either of the component running or the component as part of a composition.
Rationale: Users are used to seeing screenshots of apps in operation in various app stores, this could also be applied to services.
Category: Non-Functional – Operability – Appropriateness recognisability
Criticality: Low
Risk: Low
Associated requirements: None
Source: [Screenshot of component in composition, Screenshot of running service]

R45.5 The application should present attributes of a service that allow the user to determine its popularity (with other users).
Users should be able to determine how popular a service is with other users of the application.
Rationale: If a service is popular with other users then it is more likely to be useful to the user.
Category: Non-Functional – Operability – Appropriateness recognisability
Criticality: High
Risk: Low
Associated requirements: **R45.5**.*
Source: [Popularity], **R45.5**.*
COTS: IFTTT, AutomateIt, Yahoo! Pipes, On{X}

R45.5.1 The application should present usage statistics for services.
Services in the application should present usage statistics so that users can see how many times the service has been used, composed, downloaded, etc.
Rationale: Usage statistics can help users see how often a service has been used and hence how useful it might be to them.

Category: Non-Functional – Operability – Appropriateness recognisability
Criticality: Medium
Risk: Low
Associated requirements: None
Source: [Usage statistics], **R45.5**.*
COTS: Atooma, IFTTT, AutomateIt, Yahoo! Pipes, On{X}

R45.5.2 The application should present the number of times a service has been used.
Services in the application should indicate the number of times they have been used by all users of the application.
Rationale: If a service has been used a lot by other users then it shows it is usable and useful.
Category: Non-Functional – Operability – Appropriateness recognisability
Criticality: Medium
Risk: Low
Associated requirements: **R45.5**.*
Source: [Number of uses]
COTS: IFTTT

R45.5.3 The application should present the number of times a service has been installed.
Services in the application should indicate the number of times the service has been installed by all users of the application.
Rationale: If a service has been installed a large number of times then it is potentially more likely to be useful to the user.
Category: Non-Functional – Operability – Appropriateness recognisability
Criticality: Low
Risk: Low
Associated requirements: **R45.5**.*
Source: [Number of times installed]
COTS: Atooma, Yahoo! Pipes

R45.5.4 The application should present the number of times a service has been downloaded.
Services in the application should indicate the number of times the service has been downloaded by all users of the application.
Rationale: The number of times a service has been download is an indicator of how many other users thought it would be useful.
Category: Non-Functional – Operability – Appropriateness recognisability
Criticality: Medium
Risk: Low
Associated requirements: **R45.5**.*
Source: [Number of downloads]
COTS: Atooma, AutomateIt, On{X}

R45.5.5 The application should present the number of times a service has been downloaded vs. the number of times it has been used.
Services in the application should indicate the ratio of the number of uses of the application vs. the number of downloads.
Rationale: The ratio of uses to downloads shows how useful a service might be.
Category: Non-Functional – Operability – Appropriateness recognisability

Criticality: Low
Risk: Low
Associated requirements: **R45.5**.*
Source: [Number of downloads vs. number of uses]

R45.5.6 The application should present the users who've used the component.
Services in the application should indicate the user's who have used them, either (a) in a composite if it's a component or (b) if they've acquired it if it's a composite. Note that this could be the user's friends if connected to a Social Network.
Rationale: If the user can get an idea of who has used the component, they may be able to work out if it is applicable to them.
Category: Non-Functional – Operability – Appropriateness recognisability
Criticality: Low
Risk: Low
Associated requirements: **R45.5**.*
Source: [Who's used it]
*Dependent:***R22**

R45.5.7 The application should present ratings of that service given by other users of the application.
Services in the application should present the ratings they have been given by users of the application to show explicit popularity measure given by other users.
Rationale: If other users have rated the service highly then it is likely to be highly rated by the user.
Category: Non-Functional – Operability – Appropriateness recognisability
Criticality: Medium
Risk: Low
Associated requirements: **R45.5**.*
Source: [Ratings]
COTS: Atooma, AutomateIt, Yahoo! Pipes, On{X}

R45.5.8 The application should present reviews of that service given by other users of the application.
Services in the application should present the reviews they have been given by users of the application to show explicit popularity and opinions given by other users.
Rationale: Reviews by other users provide more information to the user on top of a rating.
Category: Non-Functional – Operability – Appropriateness recognisability
Criticality: Medium
Risk: Low
Associated requirements: **R45.5**.*
Source: [Reviews]
COTS: AutomateIt

R45.5.8.1 The application should present the number of people who found particular reviews helpful.
Reviews of services in the application should indicate the number of users who found that review helpful to show user opinions on particular reviews.
Rationale: Helpfulness of reviews allows other users to agree with a

review – adding to its credibility.
Category: Non-Functional – Operability – Appropriateness recognisability
Criticality: Low
Risk: Low
Associated requirements: **R45.5.8**
Source: [Helpful review]

R45.6 The application should present the developer/service provider of the service.
Services in the application should indicate who the developer or provider of the service is.
Rationale: If the user is familiar with a particular service provider then this may influence which service they will select.
Category: Non-Functional – Operability – Appropriateness recognisability
Criticality: High
Risk: Low
Associated requirements: **R45.6**.*
Source: **R45.6**.*
COTS: Atooma, AutomateIt, Automator, IFTTT

R45.6.1 The application should present the name of the developer/service provider.
Services in the application should show the developer/service provider by their name.
Rationale: The name of the service provider is the most obvious way to represent that service provider.
Category: Non-Functional – Operability – Appropriateness recognisability
Criticality: High
Risk: Low
Associated requirements: **R45.6**.*
Source: [Developer name]
COTS: Atooma, AutomateIt, Automator, IFTTT

R45.6.2 The application should present an icon representing the developer/service provider.
Services in the application should show the developer/service provider by their icon.
Rationale: Users are likely to associate service providers with their icon, so this should be presented to the user
Category: Non-Functional – Operability – Appropriateness recognisability
Criticality: Medium
Risk: Low
Associated requirements: **R45.6**.*
Source: [Developer icon]
COTS: Atooma, Automator, IFTTT

R45.6.3 The application should present the reputation of the developer/service provider.
Services in the application should show the reputation of the developer/service provider.
Rationale: If the user isn't already aware of the service provider, then they may need to get an idea of the reputation of that service provider.
Category: Non-Functional – Operability – Appropriateness recognisability

Criticality: Low
Risk: Low
*Associated requirements:***R45.6**.*
Source: [Developer reputation]

R45.7 The application should present Quality of Service (QoS) attributes for services.
Services in the application should present quality of service attributes such as uptime, security, etc.
Rationale: QoS attributes are used in a lot of service related applications, and could be extended to EUSC.
Category: Non-Functional – Operability – Appropriateness recognisability
Criticality: Low
Risk: Low
Associated requirements: None
Source: [QoS]
COTS: None

R46 The application should provide more information for services to which the user doesn't have access.
Rationale: Users require more information about components that they need to acquire compared with ones that they have already acquired and used.
Category: Non-Functional – Operability – Appropriateness recognisability
Criticality: Medium
Risk: Low
Associated requirements: None
Source: [More information if not owned]
COTS: IFTTT, On{X}

R47 The application should present attributes of components.
The application should present attributes of components to users. Note that these are supplementary to attributes for services.
Rationale: Attributes of components allow the user to determine information about them.
Category: Non-Functional – Operability – Appropriateness recognisability
Criticality: Critical
Risk: Low
Associated requirements: **R47**.* *Source:* **R47***
Subsequent

R47.1 The application should distinguish between input-only components and output-only components.
The application should present services that output data to the user differently from services that do not.
Rationale: Components with inputs only and outputs only should be treated differently, so should be represented differently
Category: Non-Functional – Operability – Appropriateness recognisability
Criticality: Low
Risk: Low
Associated requirements: **R47***
Source: [Input-only components vs. output-to-user components]
Subsequent

R47.2 The application should indicate if components require a data connection.

Services in the application should indicate if they require a data connection in order to execute successfully.
Rationale: Data may cost the user money, so components should indicate if they are likely to cost money.
Category: Non-Functional – Operability – Appropriateness recognisability
Criticality: Low
Risk: Low
Associated requirements: **R47**
Source: [Data connection]

R47.3 The application should indicate the location of the component's data source.
Services in the application should indicate where the data that they use comes from.
Rationale: The data may be provided in a location that is not linked with the service provider, so this should be indicated.
Category: Non-Functional – Operability – Appropriateness recognisability
Criticality: Low
Risk: Low
Associated requirements: **R47**
Source: [Component data source]

R47.4 The application should present the number of times a component has been composed.
Services in the application should indicate the number of times they have been used in a composition by all users of the application.
Rationale: the number of times it has been used in a composition is a good metric for its usefulness.
Category: Non-Functional – Operability – Appropriateness recognisability
Criticality: Medium
Risk: Low
Associated requirements: **R47**
Source: [Number of times composed]

R47.5 The application should present the cost of the component.
If use of components incurs some monetary cost, then they should indicate this.
Rationale: Users will want to know if a component costs them money.
Category: Non-Functional – Operability – Appropriateness recognisability
Criticality: High
Risk: Low
Associated requirements: **R47**
Source: [Cost]

R47.5.1 The application should indicate whether the component is free or not.
Instead of explicitly presenting the cost of the component, the application should initially just indicate whether the service will cost them money or not.
Rationale: Rather than being interested in how much a component costs, users may prefer to know whether it is free or not
Category: Non-Functional – Operability – Appropriateness recognisability
Criticality: Medium
Risk: Low
Associated requirements: **R47**
Source: [Cost (free)]

R48 The application should provide more information for composites that the user did not create.
Rationale: Users require more information for services that they have not created.
Category: Non-Functional – Operability – Appropriateness recognisability
Criticality: Medium
Risk: Low
Source: [you need more information if you don't have it]
COTS: AutomateIt, IFTTT, On{X}

R49 The application should present attributes of composites.
Rationale: Composites have attributes that users may need to know – particularly if they are acquiring these composites from a repository rather than creating them themselves.
Category: Non-functional – Operability – Appropriateness recognisability
Criticality: Critical
Risk: Low
Associated requirements: **R49**.*
Source: **R49**.*
COTS: Atooma, AutomateIt, IFTTT, On{X}, Tasker, Zapier

R49.1 Composites should be represented by the icons/representations of the components that have been coordinated together to create the composite.
Composites should use the icons of the components they are made up of in order to describe their function to the user.
Rationale: The user will be able to tell at a glance what components are within the composite.
Category: Non-functional – Operability – Appropriateness recognisability
Criticality: Low
Risk: Low
Associated requirements: **R49**
Source: [Multiple icons to show components/function]
COTS: Atooma, AutomateIt, IFTTT, On{X}

R49.2 The application should distinguish between composites the user has created and ones that they have acquired.
Users should be able to instantly tell which of the composites available to them are ones they created vs. ones that have been shared by other users and the current user has then acquired.
Rationale: Users may want to share their own composites, and they should not be able to share composites that have already been shared by someone else.
Category: Non-functional – Operability – Appropriateness recognisability
Criticality: Low
Risk: Low
Associated requirements: **R49**
Source: [My composites vs. others]
COTS: On{X}

R49.3 The application should indicate whether composites are running or enabled.
The application should indicate to users whether the application is currently executing a composite. In the case of composites that contain triggers, they should also indicate if they are enabled or disabled.
Rationale: The user needs to know which composites are running, or likely to

start running on their own, so that they know which ones are safe to edit or delete.
Category: Non-functional – Operability – Appropriateness recognisability
Criticality: Medium
Risk: Low
Associated requirements: **R49**
Source: [Running or enabled]
COTS: Atooma, Tasker, AutomateIt, Zapier, On{X}

Ease of use

R50 Composition must be simple.
The process of composition should be simple.
Rationale: Users with low technical skills should be able to use the composition application, so the process needs to be a simple one.
Category: Non-functional – Operability – Ease of use
Criticality: High
Risk: Low
Associated requirements: None
Source: [Simplicity, Boredom]
Prior identification: [Albinola et al., 2009]
COTS: Atooma, IFTTT, Zapier

R51 Descriptions of composites should be simple.
The descriptions of composites should convey the function of the composite in a plain, simple and easy-to-understand way.
Rationale: The descriptions of composites that are created by other users need to be simple so that users with low technical skills can decide whether the composite is one that they want or not.
Category: Non-Functional – Operability – Ease of use
Criticality: High
Risk: Low
Associated requirements: None
Source: [Simple composite description]

Learnability

R52 Composition must be easy to learn.
The process of composition should be easily learned by users.
Rationale: If users do not know how to use the EUSC application initially, they should be able to learn how to do so quickly so that they can create composites effectively.
Category: Non-Functional – Operability – Learnability
Criticality: Critical
Risk: Low
Associated requirements: **R53**
Source: [learnability]
Prior identification: [Albinola et al., 2009]
COTS: Atooma, IFTTT, Zapier

R53 The application could use scenarios to convey composition to users.
Since SC is something with which users aren't normally familiar, presenting introductory scenarios could motivate the use of the application.

Rationale: Scenarios are effectively stories that convey what could be possible with the application, which might inspire potential users.
Category: Non-Functional – Operability – Learnability
Criticality: Low
Risk: Low
Associated requirements: **R52**
Source: [Scenarios]

R54 The application should use metaphor to abstract the composition process.
The application should present the composition with the aid of a metaphor to abstract the composition process to something with which the user is more familiar. Potential metaphors include: Cooking – recipes in On{X} and IFTTT – jigsaw – chain – logic - physics.
Rationale: Metaphors can be used to help people associate something that is unfamiliar to them with something that is familiar.
Category: Non-Functional – Operability – Learnability
Criticality: Low
Risk: High
Associated requirements: **R54**.*
Source: [Metaphor, Jigsaw, Chain, Logic, Physics]
Prior identification: [Nestler et al., 2011]
COTS: Atooma, On{X}, IFTTT, Yahoo! Pipes

R54.1 The metaphor should be one that users agree with.
The metaphor employed by the application should be one that users agree with and can relate to.
Rationale: In order for the metaphor to be effective, the user must be able to see the analogy.
Category: Non-Functional – Operability – Learnability
Criticality: High
Risk: High
Associated requirements: **R54**.*
Source: [Disagrees with metaphor, Would prefer different metaphor]

R54.2 The metaphor should be unambiguous.
The metaphor employed by the application should be unambiguous.
Rationale: If the metaphor were ambiguous then it would be less effective in conveying EUSC to the user.
Category: Non-Functional – Operability – Learnability
Criticality: High
Risk: High
Associated requirements: **R54**.*
Source: [Ambiguity of metaphor]
COTS: IFTTT, Atooma

R55 SC applications should take inspiration from non-SC applications.
SC applications should take inspiration from other applications.
Rationale: Other applications with which users will be more familiar to introduce them to the concepts with which they may be unfamiliar.
Category: Non-Functional – Operability – Learnability
Criticality: Low
Risk: Low
Associated requirements: **R55.1**, **R56**

Source: [Take inspiration from non-SC applications]
Subsequent

R55.1 SC should be related to concepts from apps.
Rationale: Users are becoming more comfortable with apps and app stores, which are similar concepts that composition can be related to. In particular, discovering components can be likened to discovering apps in an app store, and listing components is like listing apps on a mobile phone.
Category: Non-Functional – Operability – Learnability
Criticality: Low
Risk: Low
Associated requirements: **R55**, **R56**
Source: [Choosing components is the app store, Listing composites is like listing apps]
Subsequent

R56 The application should present services with which the user is familiar.
The services that are displayed to the user should be from Internet services or similar with which the user is familiar.
Rationale: Since SC is a concept with which users are unlikely to be familiar, having services with which they are is important as it reduces the learning burden that is placed upon them.
Category: Non-Functional – Operability - Learnability
Criticality: High
Risk: Low
Associated requirements: **R55**, **R55.1**
Source: [Familiar Services]
Subsequent

Maintainability

Changeability

R57 The views of the application should be customisable.
Users should be able to customise views to change the display of, for example, component lists. It could be used to show or hide compatible components, or change how components are grouped.
Rationale: If users don't like the current representation of the composition process, they should be able to change it.
Category: Non-Functional – Maintainability - Changeability
Criticality: Low
Risk: Low
Associated requirements: None
Source: [Customisation of application views]

R58 Users should be able to manually choose icons for the composites that they have created.
Users should be able to manually select the icon to represent composites that they have created.
Rationale: Users should be able to personalise the composite that they create.
Category: Non-Functional – Maintainability - Changeability
Criticality: Low

Risk: Low
Associated requirements: None
Source: [Manually choose icons]
Subsequent

R59 The application should provide a mechanism for restricting composition.
Users of the application could be overwhelmed by the possibilities of composition and restricting this process should make this process less overwhelming.
Rationale: If composition were completely free, then there would be nothing to restrict the user, and this could be intimidating.
Category: Non-Functional – Maintainability - Changeability
Criticality: Medium
Risk: Low
Associated requirements: None
Source: [Restrictiveness in composition, Freedom of composition]
COTS: Atooma, Tasker, IFTTT, Zapier, Automator, AutomateIt

R60 Grouping should be customisable.
The method that the application uses to group services should be customisable by the user.
Rationale: Users should be able to change the grouping metric that is applied to the services to suit their needs.
Category: Non-Functional – Maintainability - Changeability
Criticality: Low
Risk: Low
Associated requirements: None
Source: [Customisation of grouping]

R61 Services should be manually grouped.
Users should be able to sort services into their own groups rather than grouping by a property of the service.
Rationale: Users may have their own groupings for services that they wish to user over those provided by the application or service developers.
Category: Non-Functional – Maintainability – Changeability
Criticality: Low
Risk: Low
Source: [Manual grouping]

Modularity

R62 Components should be simple.
Components should be presented as black-box-like entities, thus abstracting the technical details of the inner workings of the components.
Rationale: Simple components mean that the user has more control over the complexity of the composite, rather than the complexity being hidden within the components.
Category: Non-Functional – Maintainability – Modularity
Criticality: Medium
Risk: Medium
Associated requirements: None
Source: [Components as black boxes, Complexity in composition]
COTS: Atooma, Automator, AutomateIt, IFTTT, Zapier, Yahoo! Pipes, Quartz Composer

UI

R1 The application should represent components to users.
Components must be represented to users of the SC application as separate entities so that users can discover them, and coordinate them in composition.
Rationale: Users need to be able to interact with the components that they will use in the composition process.
Category: Non-functional – UI
Criticality: Critical
Risk: Low
Associated requirements: **R45**, **R47**
Source: [Components]
COTS: Atooma, AutomateIt, Automator, IFTTT, Quartz Composer, Tasker, Yahoo! Pipes

R2 Component lists should be split into pages.
When components are being presented to the user in various parts of the application, they need to be split up in order for the user to be able to process the list being presented to them.
Rationale: Having one long list of components could be intimidating to users.
Category: Non-functional – UI
Criticality: Low
Risk: Low
Associated requirements: None
Source: [Composition pages]

R3 The application should represent composites to users.
Composites must be represented to users of the SC application as separate entities so that users can discover them, and coordinate them in composition.
Rationale: Users need to be able to interact with the composites that they create or discover.
Category: Non-functional – UI
Criticality: High
Risk: Low
*Associated requirements:***R45**, **R49**
Source: [Composites]
COTS: Atooma, AutomateIt, Automator, IFTTT, On{X}, Quartz Composer, Tasker, Yahoo! Pipes

R4 The composition process should be split into pages.
As with the lists of components throughout the application, the composition process should be broken down into pages to make it easier to comprehend by the user.
Rationale: Presenting the whole composition process on a single page could overload the user.
Category: Non-functional – UI
Criticality: Medium
Risk: Low
Associated requirements: None
Source: [Composition in pages]
COTS: IFTTT

C.3 Structural

Application Structure

R5 Two application views/versions: mobile and desktop.
Different (physically) sized composition platforms allow for different potential compositions, as well as allowing for compositions in different contexts.
Rationale: Components that users want to interact with might be in different contexts, making it more intuitive to also present the composition application in these differing contexts.
Category: Structural – application structure
Criticality: Low
Risk: Medium
Associated requirements: None
Source: [Two versions: mobile and desktop, Context of composition]
Prior identification: [Namoun et al., 2010b]
COTS: On{X}
Subsequent

R6 Composites should be integrated with the OS.
Composites should be accessible directly from the platform on which they are created, without having to go through the application from which they were composed.
Rationale: It is more convenient for the user to not have to go through the EUSC application to access their composites.
Category: Structural – application structure
Criticality: Medium
Risk: Low
Associated requirements: **R7**
Source: [OS integration, File dialog]
COTS: Tasker

R7 Users should be able to access composites via desktop/home screen shortcuts or widgets.
Users should be able to create shortcuts or place widgets on the home-screen of their device through which they can invoke composites.
Rationale: It is more convenient for the user to not have to go through the EUSC application to access their composites.
Category: Structural – application structure
Criticality: Low
Risk: Low
Associated requirements:
Source: [Shortcuts to composites, Control composites via widgets]
Specific

Composition Structure

R8 The application should support different types of component.
The application should support different types of component that users might want, and ensure that these different types are conveyed to the user.
Rationale: Different types of component allow to SC application to facilitate more powerful compositions.
Category: Structural – Composition structure

Criticality: High
Risk: Low
Associated requirements: **R8.1**, **R8.2**
Source: **R8.1**, **R8.2**
COTS: Atooma, IFTTT, AutomateIt, Zapier, Yahoo! Pipes, On{X}

R8.1 The application should allow users to compose pervasive services.
The application should allow users to compose pervasive services that are present in the user's environment. For example in the home (e.g. [Bottaro et al., 2007]), or in the office.
Rationale: Pervasive components allow the user to interact with web-enabled devices in the home and environment in an Internet-of-Things manner.
Category: Structural – Composition structure
Criticality: Low
Risk: Low
Associated requirements: **R8.2**
Source: [Composing pervasive services]
Prior identification: [Bottaro et al., 2007]
COTS: IFTTT

R8.2 The application should support triggers.
The application should allow users to use triggers in the composition. Triggers are services that are activated automatically based on some external event such as receiving an SMS.
Rationale: Triggers are a powerful component that allow compositions to react to external events rather than always being triggered by the user.
Category: Structural – Composition structure
Criticality: High
Risk: Low
Associated requirements: **R8**
Source: [Triggers, Triggers vs. other components]
Prior identification: [Albreshne and Pasquier, 2011]
COTS: Atooma, IFTTT, On{X}

R9 Composition should be "infinite".
The application should allow users to take composites that have been created using the application and use them as components to be composed in a new composite.
Rationale: composition is a considerably more powerful concept if the user can re-use composites they have created in new compositions.
Category: Structural – Composition structure
Criticality: Medium
Risk: Low
Associated requirements: None
Source: [Infinite composition]

D Explicit Design Space for EUSC Applications

This appendix lists each of the design elements within our explicit design space. To describe each design element, we present the following properties:

- **Name**: The name of the design element.
- **Description**: A short description of the design element.
- **Type**: The 'type' of the design element: either category, decision, or solution.
- **Stage added**: The stage of our method at which it was added. These are specific to our method, and are one of the following:
  1. DS collation
  2. Literature review
  3. Application review (I or II)
  4. Requirements gathering
  5. Design study
- **Parent design element**: The design element that is the parent to this one, e.g. the decision to which this is the solution, the parent decision of this decision, or the category in which this decision is positioned.

The elements of the design space are presented depth first, and the properties they present are organised as follows:

Name **Type**: **Stage added** solves **Parent**
Description.

D.1 EUSC Design Space: Entity Category

Functional category: DS collation –
The DS that contains all of the functions that an EUSC tool could implement.

SC Life cycle decision: DS collation solves Functional
The different stages of Service Composition.

Inception decision: DS collation solves SC Life cycle
The stage at which the user comes up with the idea for the composite that they wish to create, or the problem that they wish to solve with composition.

Feasibility option: DS collation solves Inception
The stage at which the user assesses the feasibility of the composite that they want to create.

Project planning option: DS collation solves Inception
The stage at which the user plans what they are going to do with the tool.

Analysis decision: DS collation solves SC Life cycle
The stage at which the user analyses what they want to do and how they might want to go

about doing it.

Domain option: DS collation solves Analysis
The stage at which the user analyses the domain in which they are going to create their composite.

Activity option: DS collation solves Analysis
The stage at which the user analyses what they want their composite to do.

Specification decision: DS collation solves SC Life cycle
Where the user specifies what they want to the out of the EUSC tool.

Specification type decision: Literature review solves Specification
The "type" of specification that is created.

Needs option: DS collation solves Specification type
The stage at which the user specifies their needs to the tool.

QoS requirements option: DS collation solves Specification type
The stage at which the user specifies the "Quality of Service" (QoS) requirements that they have for the composite that they want to create. e.g. Uptime, network latency, etc.

Specification creator decision: Literature review solves Specification
The person who creates the specification of the functionality of the intended composite.

Users option: DS collation solves Specification creator
The specification of the composite is created by the users themselves.

Application developers option: DS collation solves Specification creator
The specification of the composite is created by the application developer of the SC tool.

Specification time decision: Literature review solves Specification
The time at which the specification of the functionality of the intended composite is created.

Runtime specification option: DS collation solves Specification time
The specification of the composite is created at runtime.

Development time option: DS collation solves Specification time
The specification of the composite is created at development time.

Specification method decision: Literature review solves Specification
The method by which the specification of the functionality of the intended composite is

created.

User interaction option: DS collation solves Specification method
The specification of the composite is created through direct interaction with the user.

Source code option: DS collation solves Specification method
The specification of the composite is created in the source code of the EUSC tool.

Specification Plicitness decision: Literature review solves Specification
Whether the specification of the functionality of the intended composite is implicit or explicit.

Implicit specification option: DS collation solves Specification Plicitness
The specification of the composite is created implicitly.

Explicit specification option: DS collation solves Specification Plicitness
The specification of the composite is created explicitly.

User goal/requirement mapping option: Requirements Gathering solves Specification
The system maps the user's goals and requirements to potential solutions.

No specification option: Requirements Gathering solves Specification
No mechanism for creating specifications is provided.

Realisation decision: DS collation solves SC Life cycle
The stage at which the composite is realised.

Design option: DS collation solves Realisation
The stage at which the user designs the composite that they want to create.

Construct composition decision: DS collation solves Realisation
The stage in the process where the user composes the components together.

Composition type decision: DS collation solves Construct composition
The "type" of composition that the tool supports, e.g. process, data, or interface.

Logic/process composition option: DS collation solves Composition type
The composition involves connecting up a series of processes.

Presentation/UI composition decision: DS collation solves Composition type
The tool allows the user to compose new user interfaces for composites.

UI Customisation option: DS collation solves Presentation/UI composition

The user interface of the tool can be customised.

Automatic UI Generation option: DS collation solves Presentation/UI composition
The User Interface of the composite service is generated automatically (note that this is normally only applicable in presentation-layer composition).

Selecting & customising UI option: DS collation solves Presentation/UI composition
The user can use composition to select and customise the UI of the composite.

Visual UI composition option: DS collation solves Presentation/UI composition
The user interface of the composite service is created by visually connecting components together. Note that this is only applicable to presentation-layer composition.

Textual UI composition option: DS collation solves Presentation/UI composition
The user interface of the composite service is created by manipulating text (such as using some markup language). Note that this is only applicable to presentation-layer composition.

Advanced UI generation option: DS collation solves Presentation/UI composition
The tool allows the user to create advanced user interfaces for the components that they create using it.

Data composition option: DS collation solves Composition type
The composition process involves modifying a set of data by composing services together.

Infinite composition option: Requirements Gathering solves Construct composition
Composites created by the tool can in turn be used as components in other compositions.

Composition technique decision: DS collation solves Construct composition
How the system allows the user to interact with the composition stage of the process.

Editable example option: DS collation solves Composition technique
The tool provides examples that the user can edit.

Language decision: Requirements Gathering solves Composition technique
Composition is performed using some language.

DSL decision: Requirements Gathering solves Language
The language is a domain specific language - not a full language.

Textual DSL decision: DS collation solves DSL
The interaction with the composition process is via a textual domain-specific language (a subset of a full programming language).

Textual DSL editors option: DS collation solves Textual DSL
Composition is performed through a text editor into which the user must enter a text-based Domain Specific Language (DSL).

Textual DSLs in Dialog fields option: DS collation solves Textual DSL
The user interacts with the composition by using elements from a form or dialog.

Visual DSL decision: DS collation solves DSL
The interaction with the composition process is via a graphical domain-specific language (a subset of a full programming language).

Visual data flow languages option: DS collation solves Visual DSL
Composition is performed through a visual data flow language.

Visual workflow/process oriented languages option: DS collation solves Visual DSL
Composition is performed through a visual workflow language.

Iconic option: DS collation solves Visual DSL
Composition is performed through a visual language based on icons.

Full language decision: DS collation solves Language
The method of interaction involves the user using some form of language to indirectly interact with components.

Scripting language option: DS collation solves Full language
Composition is performed by the user entering a scripting language.

Programming environment option: DS collation solves Full language
Composition is performed through a fully-fledged programming environment.

Story-based composition option: Design study solves Composition technique
The composition process leads the user through a story whilst components are added and connected in the composition.

Collaborative composition option: Design study solves Composition technique
Composition can be performed by more than one user at the same time.

Non-language decision: Design study solves Composition technique
The interaction technique to create the composition isn't based on a language.

Form-based option: DS collation solves Non-language
Composition is performed by the user interacting with a form.

Drag-and-drop option: DS collation solves Non-language
The interaction with the components and the composition allows the user to drag-and-drop the components to connect them.

PbD option: DS collation solves Non-language
The interaction with the composition is by demonstration. (For example, recording macros)

Spreadsheets option: DS collation solves Non-language
The user interacts with the composition process by manipulating a spreadsheet.

Web Page customisation option: DS collation solves Non-language
Composition is achieved by connecting a series of webpages together (note that this is only applicable to presentation-layer composition)

Dialog-based wiring of widgets option: DS collation solves Non-language
Composition is performed by the user connecting dialogs together with wires.

Live composition option: DS collation solves Non-language
Composition is performed "live", where the effects of composition are reflected immediately.

Composition editing decision: Design study solves Construct composition
The process of composition can be edited once the user has added some components to it.

Edit sections of composition decision: Tool review 2 solves Composition editing
Sections of the composition (more than one component) can be edited.

Select sections of composition option: Design study solves Edit sections of composition
Sub-sections (one or more components and their links) of the composition can be selected.

Remove sections of composition option: Design study solves Edit sections of composition
Sub-sections (one or more components and their links) of the composition can be removed.

Copy sections of composition option: Design study solves Edit sections of composition
Sub-sections (one or more components and their links) of the composition can be copied.

Paste sections of composition option: Design study solves Edit sections of composition
Sub-sections (one or more components and their links) of the composition can be pasted.

Save sections of composition option: Design study solves Edit sections of composition
Sub-sections (one or more components and their links) of the composition can be saved.

Execute sections of composition option: Design study solves Edit sections of composition
Sub-sections (one or more components and their links) of the composition can be executed.

Edit components in composition decision: Tool review 2 solves Composition editing
Components within the composition can be edited.

Select component in composition option: Tool review 2 solves Edit components in composition
A single component in the composition can be selected.

Copy component in composition option: Tool review 2 solves Edit components in composition
A single component in the composition can be copied.

Paste component in composition option: Tool review 2 solves Edit components in composition
A single component in the composition can be pasted.

Activate/deactivate component in composition option: Tool review 2 solves Edit components in composition
A component in the composite can activated or deactivated.

Insert component in composition option: Tool review 2 solves Edit components in composition
Components can be inserted into the composition at a particular position.

Discovery decision: Literature review solves Realisation
The part of the process where the user discovers services that they can use - either composites that have already been created by others, or components that they can use in composition.

Search decision: Literature review solves Discovery
Finding services by searching for them.

Text-based search option: DS collation solves Search
Discovery of components is carried out by entering some text to specify what is required.

Search metrics decision: Requirements Gathering solves Search
The different ways users can search for components.

Search by function (description) option: Requirements Gathering solves Search metrics
The user can search for components based on the function that they perform.

Search by name option: Requirements Gathering solves Search metrics
The user can search for components based on the name of the component.

Component-based search (i.e. search for Trigger or action) option: Tool review 1solves Search metrics
Searching in the tool is carried out by searching based on the components that are in the composite.

Search by technology option: Design study solves Search metrics
Search by the technology that the component or composite uses. This could be, for example, specialised hardware.

Search by effect option: Design study solves Search metrics
Search by the effect that the component or composite has on the state of the world.

Browsing decision: Literature review solves Discovery
Discovering new components by browsing through a list of the components.

Grouping/categorisation decision: Literature review solves Browsing
Grouping services together.

Browsing composites by structural properties option: DS collation solves Grouping/categorisation
The user can browse through composites based on their structural properties, i.e. the types of component that they contain, or the types of links between the components.

Simple categorisation of components option: DS collationsolves Grouping/categorisation
Components are organised into a number of simple categories.

Sub-categories option: Tool review 1 solves Grouping/categorisation
Components and composites are organised into categories and sub-categories.

Manual grouping option: Requirements Gathering solves Grouping/categorisation
The user can manually create groups for components or composites.

Customisation of grouping option: Requirements Gathering solves Grouping/categorisation
Grouping of services should be customisable by the user of the tool.

Grouping metrics decision: Requirements Gathering solves Grouping/categorisation
Different ways of grouping components together.

Group by location option: Requirements Gathering solves Grouping metrics
The user can group services by their physical location.

Group by cost option: Requirements Gathering solves Grouping metrics

The user can group services by their monetary cost.

Group by rating option: Requirements Gathering solves Grouping metrics
The user can group services by their rating.

Group by recently used option: Requirements Gathering solves Grouping metrics
The user can group services by how recently they have been executed.

Grouping by service provider option: Tool review 1 solves Grouping metrics
The user can group services by their service provider - this can be either a single user or organisation.

Group by popularity option: Tool review 1 solves Grouping metrics
Components and composites are grouped by how popular they are with users of the tool.

Group by age option: Tool review 1 solves Grouping metrics
Components and composites can be grouped by how old they are.

Group by type option: Tool review 1 solves Grouping metrics
Components can be grouped by their type (i.e. whether they are a trigger or not).

Group by function option: Requirements Gathering solves Grouping metrics
The user can group services by the function that they perform.

Group by network requirement option: Design study solves Grouping metrics
Group components or composites by their requirement of a network connection.

Group by featured/not option: Tool review 2 solves Grouping metrics
Components or composites are grouped by whether they have been "featured" by the creators of the tool or not.

Filter decision: Tool review 1 solves Browsing
Component discovery can be performed by browsing through components along with searching and filtering.

Filter by tag option: Tool review 1 solves Filter
Components and composites can be filtered based on tags that have been assigned to them.

Filter by name option: Tool review 1 solves Filter
Components and composites can be filtered based on their name.

Basic browsing (list) option: Requirements Gathering solves Browsing
Component discovery is performed by browsing through a list.

Ranking/sorting/ordering decision: Design study solves Browsing
Rank, sort, or apply some order to a list of components or composites.

Ranking alphabetically option: Design study solves Ranking/sorting/ordering
Rank components alphabetically by their name.

Rank by popularity option: Tool review 2 solves Ranking/sorting/ordering
Components or composites are ranked by how popular they are with users of the tool.

Rank by age option: Tool review 2 solves Ranking/sorting/ordering
Components are ranked by how old they are.

Suggestions decision: DS collation solves Discovery
The tool makes suggestions to the user as to which components they should use.

Context-specific suggestions option: DS collation solves Suggestions
Suggestions are provided to the user at particular times using the tool. For example if they are viewing a component it might suggest a component whose inputs match the current component's outputs.

Feature prompts option: DS collation solves Suggestions
The tool suggests features it supports that the user might want to use.

Suggestions of composites created by other users option: Tool review 1 solves Suggestions
Users are suggested that they might want to download composites that have been created by other users of the tool.

Suggestions based on matching components option: Requirements Gathering solves Suggestions
Components are recommended to users based on whether their inputs and outputs match with those of the components that they have already added to the composition.

Recommendations from friends option: Requirements Gathering solves Suggestions
The user can get recommendations of which services to use from their friends.

Featured suggestions option: Requirements Gathering solves Suggestions
Some other form of recommendations are provided.

Discoverable entities decision: Requirements Gathering solves Discovery
The different entities - either component or composite - that can be discovered in the tool.

Components discoverable option: Requirements Gathering solves Discoverable entities
Users can discover components and use them in composition.

Composites discoverable option: Requirements Gathering solves Discoverable entities
Users can discover composites that have been created by other users and then interact with them in some manner.

Contextual discovery option: Design study solves Discovery
Discover components based entirely contextually, where components or composites are suggested based on what the user is doing, where they are, etc.

Verify/Validate decision: Tool review 1 solves Realisation
The stage in the process where the user verifies that the composite they are making does what they want it to.

Evaluation decision: DS collation solves Verify/Validate
The stage at which the user evaluates the composite they are creating or have created.

Scenario option: DS collation solves Evaluation
The tool presents some scenarios that show the user how they can use composition.

Performance option: DS collation solves Evaluation
Evaluating the performance of the composite that is being created or has been created.

Usability option: DS collation solves Evaluation
The features that the tool provides to help with its usability.

Software engineering techniques decision: DS collation solves Verify/Validate
The software engineering techniques that are supported by the tool.

Debugging output option: DS collation solves Software engineering techniques
The tool provides a way for the user to debug the composition as they are creating it.

Version control option: DS collation solves Software engineering techniques
The tool provides version control.

Testing decision: DS collation solves Software engineering techniques
Testing of composites to make sure they do what the user expects.

Real-time execution option: Tool review 1 solves Testing
The composite is executed as it would normally be, but in a test environment.

Stepped execution option: Tool review 1 solves Testing
The composite executes one step at a time, where the user can see the state of the composite after each step.

Test execution option: Tool review 1 solves Testing
The composites created using the tool can be tested by executing them.

Simulate testing of triggers option: Requirements Gathering solves Testing
The conditions that activate triggers are simulated so that the user is able to test that the rest of the composite operates as it should.

Test mode for components option: Requirements Gathering solves Testing
Components all have a test mode, so that the user can test different conditions that might be unlikely to occur in real execution.

Dummy test data option: Requirements Gathering solves Testing
Testing of composites can use dummy data so that the testing process can avoid affecting the "outside world".

Change Request Management option: DS collation solves Software engineering techniques
Changes to the composite must be made only once a change request has been submitted.

Provisioning decision: DS collation solves SC Life cycle
The stage in the process where the user adds information to the composite and then makes it ready for execution.

Annotation decision: DS collation solves Provisioning
The stage at which the user adds information to the composite that they have created or are creating.

Automatic composite description generation option: Design study solves Annotation
The descriptions of composites are generated automatically by the tool based on the descriptions of the components that are contained within it.

Deployment option: DS collation solves Provisioning
The stage at which the composite is deployed so that it an then be executed or shared.

Management decision: DS collation solves SC Life cycle
The stage where the user manages the composite that they have created or are creating, usually whilst it is being executed.

Monitoring option: DS collation solves Management
The stage at which the user or the tool monitors the execution of the composite.

Adaptation option: DS collation solves Management
The stage at which the user adapts the composite that they have created.

Component management decision: Design study solves Management
The facility to manage the components that are provided by the tool.

Request components option: Design study solves Component management
The user can request new components to be added to the tool.

Notifications of new components option: Design study solves Component management
The user receives notifications when new components are added to the tool.

Update components option: Design study solves Component management
Components can be updated to later versions.

Favourite components option: Design study solves Component management
Users can "favourite" components.

Duplicate component removal option: Design study solves Component management
Duplicate components can be removed by moderators.

Composite management decision: Design study solves Management
Managing composites so that they can be changed.

Moderators for composite repository option: Design studysolves Composite management
Some users have special privileges that allow them to moderate composites that are stored in the composite repository.

Automatic composite management option: Design study solves Composite management
Composites are managed automatically by the tool.

Execution confirmation option: Tool review 2 solves Composite management
The tool confirms to the user when a composite has been executed successfully.

Deployment/execution decision: DS collation solves SC Life cycle
The stage of the process where the user executes the composite that they have made.

Code generation option: DS collation solves Deployment/execution
To output the composition, the tool generates code that can be executed elsewhere.

Compilation option: DS collation solves Deployment/execution

The tool creates a standalone application that can be executed elsewhere.

Output types decision: DS collation solves Deployment/execution
The type of object that the completed composite becomes once it has been created and exported.

Web application option: DS collation solves Output types
The tool outputs composites as Web applications.

Standalone application option: Tool review 1 solves Output types
Composites created in the tool are output as a standalone application.

File option: Tool review 1 solves Output types
Composites created in the tool are output as a file that can then be opened and interpreted by some other application.

External feed option: Tool review 1 solves Output types
Composites created in the tool are output as a feed.

Interpretation decision: Tool review 1 solves Deployment/execution
The tool interprets the output of the composition process.

Delayed execution option: Tool review 1 solves Interpretation
The composite that has been created can be executed at a later time.

Instant execution option: Tool review 1 solves Interpretation
The tool interprets the output of the composition process.

Execution timer option: Design study solves Interpretation
Composites can be executed on a timer.

Tool functions decision: Tool review 1 solves Functional
Other functions that the tool performs that don't fit within the EUSC life cycle.

View execution history option: Tool review 1 solves Tool functions
The tool indicates the status of the process of executing a composite (i.e. which component is currently being executed).

Change pricing plan option: Tool review 1 solves Tool functions
Change the pricing plan that the user has signed up to.

Browse decision: Tool review 1 solves Tool functions
Discovering new components by browsing through a list of the components.

View all available components option: Tool review 1 solves Browse
View all of the components that are available to be used in composition by the tool.

Active channels option: Tool review 1 solves Browse
Browse the channels that the user has already activated.

Add decision: Tool review 1 solves Tool functions
Design elements that can be added to the tool.

Add API option: Tool review 1 solves Add
Add a custom API to the tool that can be used as a component.

Create component option: Tool review 1 solves Add
Creation of component services and adding them to the SC tool so that they can be composed.

Provide component wrapper option: Design study solves Add
The tool provides a set of wrappers that developers can use to create their own components.

Add plug-ins option: Tool review 2 solves Add
Add plug-ins to the tool (potentially to add more components).

Composite execution decision: Tool review 1 solves Tool functions
Tool functions that affect the execution status of the composites.

Enable composite execution option: Tool review 1 solves Composite execution
Enable a component so that it can be executed.

Disable composite execution option: Tool review 1 solves Composite execution
Disable a component so that it cannot be executed.

User management decision: Design study solves Tool functions
The user's profile and other associated functions can be modified.

User profiles option: Design study solves User management
Users have profiles that can be viewed by other users of the tool.

User information decision: Tool review 2 solves User management
Information about the user that could be displayed on their profile.

User username option: Tool review 2 solves User information
The user's username.

User nickname option: Tool review 2 solves User information
The user's nickname.

User gender option: Tool review 2 solves User information
The user's gender.

User date of birth option: Tool review 2 solves User information
The user's date of birth.

User company option: Tool review 2 solves User information
The company at which the user works.

User time zone option: Tool review 2 solves User information
The time zone in which the user lives.

User points option: Tool review 2 solves User management
The number of points that the user has acquired,

Social functions decision: Tool review 1 solves User management
Functions that relate to social network functions.

Connect social network accounts option: Tool review 1 solves Social functions
Connect the tool to the user's social network accounts.

Invite friends option: Tool review 1 solves Social functions
Invite friends to use the tool.

Device management decision: Design study solves Tool functions
The user can manage the devices that can execute the composites that they have created.

Backup/Restore decision: Tool review 2 solves Device management
Backup the user's composites or restore them to a previous state.

Backup composites option: Tool review 2 solves Backup/Restore
Backup the composites that the user has created (for example to some online storage).

Restore composites option: Tool review 2 solves Backup/Restore
Restore the composites that the user has created (for example from some online storage to the user's device).

Settings decision: Tool review 2 solves Device management

The global settings for the tool.

Background notification on/off option: Tool review 2 solves Settings
Background notifications sent by the tool can be turned on or off.

Global service settings decision: Tool review 2 solves Settings
Settings that pervade across all components and composites in the tool.

Setting - location granularity option: Tool review 2 solves Global service settings
A setting to determine how granular location awareness is in services within the tool, e.g. fine grain (house) or coarse grain (country).

Setting - units option: Tool review 2 solves Global service settings
A setting to determine what type of units the tool uses, e.g. imperial vs. metric.

Setting - sampling rates option: Tool review 2 solves Global service settings
A setting to determine how often components in the tool sample hardware, e.g. the accelerometer.

Show tutorial on startup option: Tool review 2 solves Settings
Setting to enable the tool's tutorial when the tool starts.

Save data from composition option: Design study solves Tool functions
The data that is outputted from the composition can be saved and stored somewhere.

Non-service functions decision: Design study solves Tool functions
The tool provides functions that are not fully-fledged services.

Assign variables option: Tool review 2 solves Non-service functions
Assign certain values to variables so that they can be referenced by name repeatedly.

Tool state settings decision: Tool review 2 solves Tool functions
Settings that effect the state of the tool.

Tool enabled option: Tool review 2 solves Tool state settings
A setting to enable or disable the tool as a whole.

Tool enable at startup option: Tool review 2 solves Tool state settings
A setting to enable or disable the tool when the device on which it runs is turned on.

D.2 EUSC Design Space: Entity Category

Non-Functional category: DS collation –
The DS that contains all of the non-functional design elements that an EUSC tool can implement.

Representation decision: DS collation solves Non-Functional
Representation in general.

Component representation decision: Literature review solves Representation
The representation of components.

WYSIWYG option: DS collation solves Component representation
The user interacts with the composition in a What-You-See-Is-What-You-Get manner.

Form element option: Tool review 1 solves Component representation
Components are represented as elements in a form.

Icons option: Tool review 1 solves Component representation
Components are represented by their icon.

Text option: Tool review 1 solves Component representation
Components are represented by a piece of text.

Flow diagram component option: Tool review 1 solves Component representation
Components are represented as if they are entities in a flow diagram.

Composition representation decision: Tool review 1 solves Representation
The representation of the composition step.

Wire paradigm option: DS collation solves Composition representation
Components can be connected together in composition using wires.

Spreadsheet representation option: DS collation solves Composition representation
The composition process within the tool is represented by way of a spreadsheet.

Form option: Tool review 1 solves Composition representation
Composition is represented as a form that the user fills in to perform the composition.

Flow diagram option: Tool review 1 solves Composition representation
Composition is represented as if it were a flow diagram.

Visual Language representation decision: DS collationsolves Composition representation
The representation of the composition process is a visual (non-text) domain-specific language.

Visual data flow languages representation option: DS collation solves Visual Language representation
The composition process is represented using a visual data flow language.

Visual workflow/process oriented languages representation option: DS collation solves Visual Language representation
The composition process is represented using a visual workflow language.

Textual DSL representation decision: DS collation solves Composition representation
The representation of the composition process is a text-based Domain-specific language.

Textual DSL in text editor option: DS collation solves Textual DSL representation
The composition process is represented as a text-based Domain Specific Language (DSL) in a text editor.

Textual DSLs as dialog fields option: DS collation solves Textual DSL representation
The composition process is represented as a text-based Domain Specific Language (DSL) in the form of a number of dialogs.

Full language representation decision: DS collation solves Composition representation
The representation of the composition process is a full language - usually a programming language.

Script/language-based option: DS collation solves Full language representation
Composition is performed using a scripting or programming language.

Component links decision: Tool review 1 solves Composition representation
The representation of the links between the components.

Textual links option: Tool review 1 solves Component links
The links between components are based on text rather than graphics.

Graphical links decision: Tool review 1 solves Component links
The links between components are graphical, such as icons or lines, etc.

Wire links option: Tool review 1 solves Graphical links
The connections between the components in the composition are represented as wires.

Non-wire links option: Tool review 1 solves Graphical links
The connections between components are represented as graphics, but are not wires.

No link option: Requirements Gathering solves Component links
There are no links between components in the composition.

Description of links option: Design study solves Component links
The links between components have descriptions to indicate what is happening to the user.

Composition roadmap option: Design study solves Composition representation
The tool shows a roadmap indicating the current state of the composition, and the position at which the user is in this process.

Abstraction decision: Requirements Gathering solves Representation
The abstraction employed by the tool to represent composition to the user.

Metaphor decision: Requirements Gathering solves Abstraction
The tool employs some metaphor to abstract the tool to its users.

Jigsaw metaphor option: Design study solves Metaphor
Composition is abstracted by a jigsaw-based metaphor.

Abstraction level decision: Requirements Gathering solves Abstraction
The level to which the tool abstracts composition away from its underlying technicalities.

High abstraction option: Requirements Gathering solves Abstraction level
The composition process is abstracted to a high level, relating it to something that is not closely linked to composition.

Medium abstraction option: Requirements Gathering solves Abstraction level
The composition process has some abstraction, but still resembles composition.

Low abstraction option: Requirements Gathering solves Abstraction level
The composition process is not abstracted very much.

Relate to other concepts option: Design study solves Abstraction
Composition is abstracted to other concepts with which the user is more likely to be familiar.

Community features decision: DS collation solves Non-Functional
The community of users of the EUSC tool.

Tagging option: DS collation solves Community features
Tagging of composites instead of putting them in groups

Rating option: DS collation solves Community features
The user is able to rate components.

Social networks option: DS collation solves Community features
The tool connects to users' social networks.

Collaboration decision: DS collation solves Community features
How the EUSC tool supports collaboration between different users.

Blackboard option: DS collation solves Collaboration
Interaction is in the form of a blackboard - several users can edit the composition on a single "blackboard"

Fork & Edit option: DS collation solves Collaboration
The tool allows users to fork existing composites in order to acquire their own copy which they can then edit.

Wiki option: DS collation solves Collaboration
The system provides a wiki where the user can get assistance with the tool.

Sharable entities decision: Tool review 1 solves Community features
The entities that can be shared with other users of the tool.

Composites sharable option: Requirements Gathering solves Sharable entities
Composites can be shared with other users of the tool.

Components sharable option: Requirements Gathering solves Sharable entities
The system allows the user to share components.

Forums decision: Tool review 1 solves Community features
There is a forum associated with the tool where users can get feedback from employees and other users of the tool.

Community forums option: DS collation solves Forums
The tool provides forums where users can discuss the tool and other aspects of the composition.

Artefact-centred discussion option: DS collation solves Forums
Forums allow users to have discussions that are based on the composition tool.

Online Community decision: DS collation solves Community features
The level of access that is provided to the online community of the tool.

Private option: DS collation solves Online Community
The online portion of the tool is private and is only accessible to a particular group of people, for example employees of a particular company.

Public option: DS collation solves Online Community
The online portion of the tool is publicly accessible - anyone can join.

Usability features decision: DS collation solves Non-Functional
The usability features that are supported by the tool.

Learning curve decision: DS collation solves Usability features
How steep the learning curve is for new users learning to use the tool.

Incremental programming paradigms option: DS collation solves Learning curve
Programming techniques are introduced incrementally.

Learning curve depth decision: Literature review solves Learning curve
The depth/steepness of the learning curve that is associated with using the tool.

Low option: DS collation solves Learning curve depth
The learning curve is very shallow.

Low-medium option: DS collation solves Learning curve depth
The learning curve is relatively shallow.

Medium option: DS collation solves Learning curve depth
The learning curve is medium

Medium-high option: DS collation solves Learning curve depth
The learning cure is relatively steep.

High option: DS collation solves Learning curve depth
The learning curve is very steep.

Learning support decision: DS collation solves Usability features
The support that the tool provides for the user to learn how to use it and to learn how to perform composition.

Discussion forums option: DS collation solves Learning support
Discussion forums are provided where users can discuss problems, ideas, etc.

Examples decision: Tool review 1 solves Learning support
The system provides examples of potential composites to users.

Initial examples option: Design study solves Examples
The tool comes with a set of initial examples that the user can use, modify, etc.

Incompatibility prompts option: DS collation solves Learning support
The tool highlights incompatible components to the user.

External help systems decision: Literature review solves Learning support
Help systems provided that are external to the tool, for example on an accompanying website.

Help forum/group option: Tool review 1 solves External help systems
The tool has an associated forum or group where users can go to get help.

FAQ option: Tool review 1 solves External help systems
The service composition tool has an external help or Frequently Asked Questions (FAQ) - usually on a website.

Blog option: Tool review 1 solves External help systems
The service composition tool is linked to a blog.

Video tutorials option: Tool review 1 solves External help systems
The tool provides screencasts to show users and potential users what is possible with the tool.

Offline/internal help decision: Design study solves Learning support
The tool provides a set of help that can be accessed offline, or within the tool.

Tutorials option: DS collation solves Offline/internal help
The tool provides tutorials for its users.

Help/documentation option: Literature review solves Offline/internal help
The system provides help to the user whilst they are performing composition.

API documentation option: Literature review solves Offline/internal help
Documentation is provided about the interfaces of each of the components.

Send query to developer option: Tool review 2 solves Offline/internal help
The user can send questions to the developer of the tool or component.

Error recognition decision: Design study solves Usability features
The tool recognises when errors have occurred.

Composition logic checker option: Design study solves Error recognition

The tool checks the logic of the composition to ensure that it is consistent.

Warnings option: Design study solves Error recognition
The tool provides warnings when the creates actions in the composition process that might not perform in the way the user expects.

Bug reporting option: Design study solves Error recognition
The user can report bugs that they find in the tool.

Failsafe for errors option: Design study solves Error recognition
The user can specify a failsafe that the tool can perform if an error occurs.

Export log file option: Tool review 2 solves Error recognition
The tool allows the user to export the log file that records the state of execution of composites.

User decision: DS collation solves Non-Functional
The expertise that the user of the tool is required to have before they should be able to use the EUSC tool.

Target user group decision: DS collation solves User
The target user of the EUSC tool.

Enterprise oriented option: DS collation solves Target user group
The target user of the tool is a member of an enterprise organisation.

Consumer oriented option: DS collation solves Target user group
The target user is a normal consumer, or member of the public.

User types decision: DS collation solves User
The technical expertise of the target user.

Casual users option: DS collation solves User types
The user doesn't have any knowledge of the domain in which the composition is taking place.

Power users option: DS collation solves User types
The tool is aimed at power users who are very familiar with technology.

Developers option: DS collation solves User types
The target user is required to be a technical expert.

Domain expert option: Tool review 1 solves User types
The target user is required to be an expert in the domain of the composite service.

Programming skill requirement decision: DS collation solves User
The amount of previous programming experience the target user of the tool should have.

Non-programmer option: DS collation solves Programming skill requirement
The user may or may have technical knowledge, but they have no programming knowledge.

Average option: DS collation solves Programming skill requirement
The tool is aimed at "average" users.

Expert option: DS collation solves Programming skill requirement
The user is a programmer.

Terminology decision: Design study solves User
The terminology associated with the tool and the composition process.

Domain Specificity decision: Literature review solves Non-Functional
How versatile the tool is in terms of the domain in which it operates.

Generic option: DS collation solves Domain Specificity
The tool works across multiple domains and contexts.

Specialised decision: DS collation solves Domain Specificity
The tool works in a single context (e.g. mobile) and a single domain (e.g. travel)

Mobile domain option: Tool review 1 solves Specialised
The tool operates in the mobile domain.

Web-services decision: Tool review 1 solves Specialised
The tool operates in the Web Services (WS-*) domain.

SNS option: Tool review 1 solves Web-services
The tool supports interaction with Social Networking services.

Utilities decision: Tool review 1 solves Web-services
The tool operates in the Utilities domain.

Weather option: Tool review 1 solves Utilities
The tool operates in the weather domain.

Calendar option: Tool review 1 solves Utilities
The tool operates in the calendar domain.

Location option: Tool review 1 solves Utilities
The tool operates in the location domain.

News option: Tool review 1 solves Utilities
The tool operates in the news domain.

Notes option: Tool review 1 solves Utilities
The tool operates in the notes and note-taking domain.

Business services option: Tool review 1 solves Web-services
The tool operates in the business service domain.

RSS domain option: Tool review 1 solves Web-services
The tool operates in the RSS Domain.

Chat option: Tool review 1 solves Web-services
The tool operates in the chat domain.

Home automation option: Tool review 1 solves Specialised
The tool operates in the home automation domain.

Desktop multimedia services option: Tool review 1 solves Specialised
The tool operates in the desktop multimedia services domain.

Context specific option: Requirements Gathering solves Specialised
The tool works in multiple domains, but is specific to one context. For example, a mobile tool that can create composites in multiple domains.

Computer algebra domain option: Design study solves Specialised
The chosen domain relates to computer algebra.

Flow plicitness decision: Tool review 1 solves Non-Functional
The use of flow in composition.

Control flow Plicitness decision: DS collation solves Flow plicitness
The representation of the passing of control between services (i.e. ordering)

Implicit control flow option: DS collation solves Control flow Plicitness
Control flow - the passing of control between the components - isn't represented explicitly.

Explicit control flow option: DS collation solves Control flow Plicitness

Control flow - the passing of control between the components in the composition - is represented explicitly.

Data flow plicitness decision: Tool review 1 solves Flow plicitness
The representation of the passing of data between components.

Implicit data flow option: Tool review 1 solves Data flow plicitness
The passing of data between components is represented implicitly.

Explicit data flow option: Tool review 1 solves Data flow plicitness
The passing of data between components is represented explicitly.

Security decision: Design study solves Non-Functional
Aspects of security in the tool.

Privacy decision: Design study solves Security
Aspects of privacy in the tool.

Privacy policy option: Tool review 2 solves Privacy
The privacy policy of the tool.

Lock code option: Tool review 2 solves Security
The tool has a lock code so that the user can password protect their composites.

Aesthetics decision: Requirements Gathering solves Non-Functional
The general aesthetics of the tool.

Colour option: Requirements Gathering solves Aesthetics
The use of colour in the tool.

Visual option: Requirements Gathering solves Aesthetics
The composition process should be very visual.

D.3 EUSC Design Space: Structural Category

Structural category: DS collation –
The structural DS contains all of the design elements that relate to the potential architectural design decisions.

Composition structure decision: Literature review solves Structural
Features of the composition structure.

Automation degree decision: DS collation solves Composition structure
The level of automation that the tool supports - how much influence the user has on the process vs. the tool.

Full-automation option: DS collation solves Automation degree
The tool provides fully-automated composition: the user provides a specification and the tool performs composition for them to give the desired output.

Semi-automation option: DS collation solves Automation degree
The tool provides semi-automated composition - some elements of composition are performed automatically, but other elements are manual. For example, inputs and outputs could be matched up based on their data type.

Manual creation option: DS collation solves Automation degree
The tool provides no automation - the user must perform all composition tasks manually.

Composition Layer decision: Literature review solves Composition structure
The "layer" at which composition operates - i.e. the part of the component that is connected together - it's data, it's interface, etc.

Service Layer option: Literature review solves Composition Layer
Composition at the service layer means that components are connected directly to one another, without going through any other entity.

Application Layer option: Literature review solves Composition Layer
Composition at the application layer means that each component is connected to the tool/application that performs the composition (which is then in turn connected to the next component)

Presentation Layer option: Literature review solves Composition Layer
Composition at the presentation layer means that connections are made directly between elements of each component's interface, without any additional processing.

Liveness decision: DS collation solves Composition structure
The level of "liveness" that is supported in a flow diagram representation of composition.

Level 1 option: DS collation solves Liveness
Liveness level 1 means that the representation of the composition is not linked with any sort of runtime system. The output of a tool at level 1 can only be in the form of an interface. Tools at this level are very simple because they only allow the creation of a non-executable prototype, but obviously this makes them much less useful.

Level 2 option: DS collation solves Liveness
Liveness level 2 means that the flowchart representation contains enough semantics to allow the tool to export the composition to some other tool that can execute the resulting composite.

Level 3 option: DS collation solves Liveness
Liveness level 3 means that the output of the composition process can be executed automatically, either in response to an edit or the user indicating they wish to execute the composite.

Level 4 option: DS collation solves Liveness
Liveness level 4 allows real-time editing, meaning that the composite can be edited while it is being executed.

Internal component model decision: DS collation solves Composition structure
The method by which components are represented inside the EUSC tool.

Native model option: DS collation solves Internal component model
Components are described using the native language in which they are implemented.

Abstract model option: DS collation solves Internal component model
Descriptions of components are abstracted into a different description language than the language in which they are implemented.

Composition logic decision: DS collation solves Composition structure
The logic and structures that are supported in the composition process in the tool.

Flow decision: Requirements Gathering solves Composition logic
Whether flow is present in the tool

Control flow option: DS collation solves Flow
Control flow means that control passes from one component to another in some order.

Data flow option: DS collation solves Flow
Connections between the components in the composition process pass data between the components.

Logical/programming structures decision: Requirements Gathering solves Composition

logic
The structures normally used in programming languages that the tool supports.

Logical operators decision: Requirements Gathering solves Logical/programming structures
The logical connections between components that the tool supports (e.g. and, or, not)

Logical AND option: Tool review 1 solves Logical operators
The tool supports logical AND, e.g. executing one service AND another

Logical OR option: Requirements Gathering solves Logical operators
The tool allows the user to create composites that activate if one event OR another occurs.

Logical NOT option: Requirements Gathering solves Logical operators
The tool supports logical not, e.g. if this and NOT this.

Logical EXCEPT WHEN option: Design study solves Logical operators
The tool facilitates logical EXCEPT WHEN, i.e. do this EXCEPT WHEN this happens.

Loops option: Tool review 1 solves Logical/programming structures
Looping is allowed in composition - selections of components can be executed iteratively.

Event-based option: DS collation solves Logical/programming structures
Composites created in the composition tool execute in response to events.

Branches option: Requirements Gathering solves Logical/programming structures
Composition allows branching - when one component finished, two or more others can start in parallel.

Data manipulation between components option: Design study solves Composition logic
The tool allows the user to specify how data can be manipulated by the tool when it is passed between components.

Component execution output filter option: Design study solves Composition logic
The tool can continue or stop the execution of the composite based on the output of one of the components in the composite.

Layout logic decision: DS collation solves Composition structure
The logic of the layout or organisation of the components in the composition.

Custom layouts option: DS collation solves Layout logic
The tool allows the user to create custom layouts in the composition.

Screen/page flow option: DS collation solves Layout logic
The composition process has a series of pages.

Text editor option: Tool review 1 solves Layout logic
The user interacts with the composition process in a text editor.

Templates decision: Tool review 1 solves Layout logic
The use of templates in the tool that the user can use to make composition easier.

Template level decision: Tool review 1 solves Templates
The level at which the template is applied.

Tool-level template option: Tool review 1 solves Template level
The template is at the tool-level - all composites created with the tool must follow the same template.

Composition-level template option: Tool review 1 solves Template level
The template is at the composition-level, the user can selected different templates to follow when performing composition.

Template type decision: Requirements Gathering solves Templates
The type of structure that the template provides, i.e. services must be executed linearly, or in parallel, or in response to events.

Linear template option: Requirements Gathering solves Template type
The tool employs a linear template - services are executed one after the other.

No template option: Requirements Gathering solves Templates
The tool doesn't use any templates - the user can configure the components as much as they want.

Single page/canvas option: Design study solves Layout logic
Composition is performed on a single page or canvas.

Tool structure decision: Literature review solves Structural
How the composition stage itself is structured.

Tool type decision: DS collation solves Tool structure
The type of application that the tool is.

Framework option: DS collation solves Tool type
The tool is a framework.

Platform option: DS collation solves Tool type
The tool is within a platform or framework.

Minimum device for framework decision: DS collation solves Tool type
The minimum type of device that is required to run the framework (assuming the EUSC tool is a framework)

Server (framework) option: DS collation solves Minimum device for framework
A server is the minimum device required to run the EUSC tool framework.

PDA (framework) option: DS collation solves Minimum device for framework
A PDA is the minimum device required to run the EUSC tool framework.

J2SE (framework) option: DS collation solves Minimum device for framework
A device that supports J2SE is required to run the EUSC tool framework.

Minimum device for service decision: DS collation solves Tool type
The minimum device that is required to run services created using the tool.

PDA (service) option: DS collation solves Minimum device for service
A PDA is the minimum device required to run composites that are created using the tool.

Mote (service) option: DS collation solves Minimum device for service
A mote is the minimum device required to run composites that are created using the tool.

Platform choice decision: DS collation solves Tool type
The location of the tool.

Desktop option: DS collation solves Platform choice
The tool operates as a desktop application.

Mobile option: Tool review 1 solves Platform choice
The composition tool operates on mobile.

Tablet option: Tool review 1 solves Platform choice
The tool runs on tablets.

Web option: Tool review 1 solves Platform choice
The tool operates on the Web.

Application option: Tool review 1 solves Tool type
The service composition tool is a standalone application.

Personal web portal option: DS collation solves Tool type
The tool is a web portal that the user can personalise.

Web page option: Tool review 1 solves Tool type
The service composition tool is a Web page or web app.

Virtual browser option: DS collation solves Tool type
The tool is a Web browser.

Plug-in decision: Requirements Gathering solves Tool type
The service composition tool is a plug-in to another application.

Browser based tools widgets option: DS collation solves Plug-in
The tool is a widget that is provided within a Web browser.

Toolbars option: DS collation solves Plug-in
The tool is a toolbar.

Extension client to a Web browser. option: DS collation solves Plug-in
The tool is a browser extension.

Data sources for components decision: Literature review solves Tool structure
The source from which the component acquires their data.

Data sources decision: DS collation solves Data sources for components
The data sources of the components.

One simple source option: DS collation solves Data sources
Components have one simple data source.

Two or more sources option: DS collation solves Data sources
Components have two or more data sources.

One source with a form option: DS collation solves Data sources
Components have one data source with a form that is used to interact with it.

Combining two or more using DB join option: DS collation solves Data sources
Components have multiple data sources and are connected together using a database join.

Data retrieval strategy decision: DS collation solves Data sources for components
The method that is used by components to retrieve data from their data source.

DOM option: DS collation solves Data retrieval strategy
Components get their data by scraping Web pages.

Widget option: DS collation solves Data retrieval strategy
Components get their data via various widgets.

RDF option: DS collation solves Data retrieval strategy
The tool retrieves its data using RDF.

Data message format decision: Literature review solves Data sources for components
The data format that is used to pass messages between components.

XML option: DS collation solves Data message format
Services are described using plain XML - eXtensible Markup Language.

JSON option: DS collation solves Data message format
Services are described using JSON - JavaScript Object Notation. http://www.json.org/

Parameter-value pairs option: DS collation solves Data message format
The data that is passed between components is in the form of parameter- or name-value pairs.

Lightweight data format option: Design study solves Data message format
The format of the data sent between the components is lightweight.

Data integration decision: DS collation solves Data sources for components
The method by which the tool integrates the data from the components.

Manual data integration option: DS collation solves Data integration
The data from the components is integrated manually.

Join only option: DS collation solves Data integration
The data from the components is integrated using a database join.

Union only option: DS collation solves Data integration
The data from the components is integrated using a set union.

Widgets option: DS collation solves Data integration
The data from components is integrated using widgets.

Architecture decision: DS collation solves Tool structure
The architecture of the tool and how it connects with others (if at all).

Client option: DS collation solves Architecture
Composition is performed on a client application that may or may not connect to some sort of server.

Server option: DS collation solves Architecture
Composition is performed on a server that the user connects to through some client.

Infrastructure decision: DS collation solves Tool structure
The infrastructure that is required for the tool to operate.

Fixed option: DS collation solves Infrastructure
The infrastructure on which the tool operates is fixed.

Ad hoc option: DS collation solves Infrastructure
There are no fixed elements in the infrastructure that the tool requires

Topology decision: DS collation solves Tool structure
The topology of the different instances of the tool.

Centralised option: DS collation solves Topology
The topology of the network is centralised - there is some central element that the tool uses (e.g. a repository)

Decentralised option: DS collation solves Topology
The topology of the network is decentralised - any communication operates directly between peers.

Reuse decision: DS collation solves Tool structure
The support for re-use of composites or components that have been created by other users of the tool.

Component repository option: DS collation solves Reuse
The tool has a repository of components that can be discovered in the discovery stage and then composed.

Composite repository option: DS collation solves Reuse
Users can perform composition by acquiring examples (either those created by the system creator, or other others), and then modifies those examples.

Automatic reuse of data extractors option: DS collation solves Reuse
The data extractors that gather the data from the components' data sources are re-used automatically by other components.

Extensions decision: DS collation solves Tool structure
The extensions that can be applied to the tool or the services that it provides.

Extension APIs option: DS collation solves Extensions
The tool allows the use of extension APIs to extend the functionality provided by the services.

System integration option: Requirements Gathering solves Extensions
Integrating the tool with the OS on which it is running.

Contingency management decision: DS collation solves Tool structure
How the tool deals with problems that arise during either the composition process or the execution of the composite that has been created.

Automatic option: DS collation solves Contingency management
Problems are detected and dealt with automatically.

None option: DS collation solves Contingency management
Problems are not detected.

Runtime option: DS collation solves Contingency management
Problems are detected and dealt with at runtime.

Detection option: DS collation solves Contingency management
Problems are detected but not dealt with.

Execution context decision: Tool review 1 solves Tool structure
The context (i.e. mobile or fixed/desktop) that the composite operates in when it is executed.

Mobile execution context option: Tool review 1 solves Execution context
The service composition tool operates in a mobile context (on a mobile phone, tablet, or other similar device).

Web execution context option: Tool review 1 solves Execution context
The service composition tool operates in a Web context, i.e. it is accessible through a web browser (this browser can operate in any context).

Desktop execution context option: Tool review 1 solves Execution context
The service composition tool operates in a desktop context (on a desktop PC/Mac)

Multiple versions of tool decision: Design study solves Tool structure
There are multiple versions of the tool, e.g. a Web version and a mobile version.

Multiple interaction techniques option: Design study solves Multiple versions of tool
The tool provides multiple different ways that the user can perform composition with the tool.

Multiple representations option: Design study solves Multiple versions of tool
The tool provides multiple different representations of the composition process.

Customisable service representations option: Design study solves Multiple versions of tool
The representation of service components can be customised by the user.

Technologies decision: Literature review solves Structural
The technologies that the tool uses to support composition.

Component Communication protocol decision: Literature review solves Technologies
The protocol that the tool uses to allow the components in the composition to coordinate and pass data between one another.

SOAP option: DS collation solves Component Communication protocol
Messages are passed between each other using SOAP - the Simple Object Access Protocol: http://www.w3.org/TR/soap/

REST option: DS collation solves Component Communication protocol
The components use REST - REpresentational State Transfer (but can send/receive any data format) http://en.wikipedia.org/wiki/Representational_state_transfer

Syndication format decision: DS collation solves Technologies
The syndication format that is used to pass data in components (i.e. RSS/ATOM)

RSS option: DS collation solves Syndication format
The syndication used is RSS - Really Simple Syndication. (This is where the component gets its data from)

ATOM option: DS collation solves Syndication format
The syndication format used is ATOM (http://www.ietf.org/rfc/rfc4287)

Implementation language decision: DS collation solves Technologies
The programming language in which the tool is implemented.

JavaScript option: DS collation solves Implementation language
The tool is implemented in JavaScript.

Android Dalvik Java option: Tool review 1 solves Implementation language
The tool is written in Android

Description language decision: Literature review solves Technologies
The language with which the components are described.

WSDL option: DS collation solves Description language
Services are described using the Web Services Description Language - WSDL. http://www.w3.org/TR/wsdl

WADL option: DS collation solves Description language
Services are described using the Web Application Description Language (WADL)

Proprietary description option: DS collation solves Description language
The description of the component is in a proprietary language.

Component types decision: Tool review 1 solves Structural
The types of component that are supported by the tool.

Triggers option: Tool review 1 solves Component types
Triggers are services that are executed when some external condition is met, rather than being executed by the tool itself. For example a trigger might be executed when a SMS is received.

Actions option: Tool review 1 solves Component types
An action is a component that is activated when it is called by the tool.

Component Inputs and outputs decision: Tool review 1 solves Component types
The inputs to and outputs from components (the data that they pass to one another).

Single composite option: Tool review 1 solves Component Inputs and outputs
The inputs and outputs of the components are a complex object, rather than being a collection of simple items (e.g. a phone contact object rather than name, phone number, etc.)

Multiple atomic option: Tool review 1 solves Component Inputs and outputs
The inputs and outputs of components are several atomic objects that, rather than being a simple complex object (e.g. a contact's name and number as two separate I/Os rather than a single "phone contact" object).

Input, output data types option: Requirements Gathering solves Component Inputs and outputs
The data types of the inputs and outputs of the components in the composition.

Recurring event option: Tool review 1 solves Component types
The tool supports events that occur frequently.

Pervasive components option: Requirements Gathering solves Component types Services in the environment (e.g. projectors) should be supported.

D.4 EUSC Design Space: Entity Category

Entity category: Requirements Gathering –
The DS that contains all of the interactions with the various entities in composition, and their attributes.

Service attributes decision: Tool review 1 solves Entity
Attributes of services that are presented to the user of the tool

Composite attributes decision: Tool review 1 solves Service attributes
Attributes of composites that are presented in the tool.

Composite components contained decision: Tool review 1 solves Composite attributes
The components that are contained within the composite.

Contained component names option: Tool review 1 solves Composite components contained
Composites present the names of the components that make them up.

Contained component categories option: Tool review 1 solves Composite components contained
The categories of the components that are contained within the composite.

Contained component category icons option: Tool review 1solves Composite components contained
The icons of the categories of the components that are contained within the composite.

Contained component descriptions option: Tool review 1 solves Composite components contained
Composites present the descriptions of the components that make them up.

Composite functional attributes decision: Requirements Gathering solves Composite attributes
Attributes of composites that help users determine the function of that composite.

Composite icon option: Tool review 1 solves Composite functional attributes
Composites present their icon.

Composite name option: Tool review 1 solves Composite functional attributes
Composites in the tool present their name.

Composite creator option: Tool review 1 solves Composite functional attributes
Composites present their creator.

Composite category option: Tool review 1 solves Composite functional attributes
Composites present the category in which they fit.

Composite activation status option: Tool review 1 solves Composite functional attributes
The activation status of the component (i.e. whether it is currently active or not).

Composite description option: Requirements Gathering solves Composite functional attributes
Composites in the tool present a description of what they do.

Composite activation conditions option: Design study solves Composite functional attributes
The list of conditions that need to be met for the component to be activated.

Execution order option: Tool review 1 solves Composite functional attributes
The composite identifies the order in which the components within it are executed.

Composite popularity attributes decision: Requirements Gathering solves Composite attributes
Attributes of composites that users can use to determine how popular they are.

Composite num times used option: Tool review 1 solves Composite popularity attributes
Composites in the tool present the number of times they have been used/executed.

Composite ratings option: Tool review 1 solves Composite popularity attributes
Composites present the rating that they have been given by users of the tool.

Composite num ratings option: Tool review 1 solves Composite popularity attributes
Composites present the number of times they have been rated.

Composite num downloads option: Tool review 1 solves Composite popularity attributes
The composite indicates the number of times that it has been downloaded by users of the tool.

Composite num clones option: Tool review 1 solves Composite popularity attributes
The number of times the composite has been cloned.

Composite reviews option: Tool review 1 solves Composite popularity attributes
Composites present the reviews that they have been given by other users.

Composite num SNS shares option: Requirements Gatheringsolves Composite popularity attributes
Composites in the tool present the number of times they have been shared on a Social

Networking service.

Composite num users option: Requirements Gathering solves Composite popularity attributes
Composites present the number of times that they have been used.

Composite num reviews option: Requirements Gathering solves Composite popularity attributes
The number of reviews that a composite has been given.

Composite starred option: Tool review 1 solves Composite popularity attributes
Whether the composite has been starred or not.

Composite tags option: Tool review 1 solves Composite popularity attributes
Composites present tags to denote some feature that they have to the user.

Composite other attributes decision: Requirements Gatheringsolves Composite attributes
Attributes of composites that are neither functional or popularity-based.

Composite age option: Tool review 1 solves Composite other attributes
Composites in the tool present when they were created, or how long since they were created.

Composite copyright option: Requirements Gathering solves Composite other attributes
The details of the owner of the copyright of the composite.

Composite notes option: Requirements Gathering solves Composite other attributes
Any notes associated with the composite.

Composite version option: Requirements Gathering solves Composite other attributes
The version number of the composite.

Component attributes decision: Tool review 1 solves Service attributes
Attributes of components that are presented to users of the tool.

Component functional attributes decision: Requirements Gathering solves Component attributes
Attributes of components that are presented to users so that they can determine its function.

Component icon option: Tool review 1 solves Component functional attributes
Components present their icon.

Component name option: Tool review 1 solves Component functional attributes
Components present their name.

Component description option: Tool review 1 solves Component functional attributes
Components present a description of what they do.

Component inputs option: Requirements Gatheringsolves Component functional attributes
Components present their inputs.

Component outputs option: Requirements Gathering solves Component functional attributes
Components present their outputs.

Component results option: Requirements Gatheringsolves Component functional attributes
The results that executing the component has on the state of the world, that would not be considered as outputs,

Component preconditions option: Requirements Gathering solves Component functional attributes
Components present the requirements/pre-requisites that need to be met before they can be executed successfully.

Component example usage option: Tool review 1 solves Component functional attributes
Components present an example of how they can be used.

Component tags option: Tool review 1 solves Component functional attributes
Components present "tags" to show various attributes.

Component category option: Requirements Gathering solves Component functional attributes
Components present their category.

Component creator/provider option: Requirements Gathering solves Component functional attributes
Components present the identity of the person/organisation that created them.

Component related actions option: Requirements Gatheringsolves Component functional attributes
Components present a list of related actions/components.

Component parameters decision: Requirements Gathering solves Component functional attributes
The parameters that can be passed to the component to change how it operates.

Component parameters decision: Requirements Gathering solves Component parameters

The parameters that can be passed to the component to change how it operates.

Component parameter description option: Tool review 1 solves Component parameters
A description of the component's parameters.

Component default parameters option: Requirements Gathering solves Component parameters
The composite has a set of default parameters that are used if the user does not specify any other parameters to use.

Component parameter name option: Requirements Gathering solves Component parameters
The name of the parameter that can be passed to the component.

Component popularity attributes decision: Requirements Gathering solves Component attributes
Attributes of components that allow users to assess the popularity of a component.

Component num uses option: Tool review 1 solves Component popularity attributes
Components present the number of times they have been used.

Component num SNS shares option: Tool review 1solves Component popularity attributes
Components present the number of times they have been shared on Social Networking services.

Component num downloads option: Tool review 1solves Component popularity attributes
Components present the number of times they have been downloaded.

Component num users option: Requirements Gathering solves Component popularity attributes
Components present the number of times they have been used.

Component reviews option: Tool review 1 solves Component popularity attributes
Components present the reviews that they have been given by other users of the tool.

Component num reviews option: Requirements Gathering solves Component popularity attributes
The number of reviews that a component has been given.

Component ratings option: Tool review 1 solves Component popularity attributes
Components present the ratings they have been given by other users of the tool.

Component num ratings option: Requirements Gathering solves Component popularity

attributes
Components present the number of times they have been rated.

Component other attributes decision: Requirements Gathering solves Component attributes
Attributes of components that neither present function or popularity.

Component copyright option: Tool review 1 solves Component other attributes
Components present their copyright information.

Component location option: Tool review 1 solves Component other attributes
Components present their physical location (i.e. whether they are available online, on the device, etc.)

Component version option: Tool review 1 solves Component other attributes
Components present their version number.

Component age option: Tool review 1 solves Component other attributes
Components present either their age, or the date at which they were created.

Service Interactions decision: Tool review 1 solves Entity
Interactions that users can have with services through the tool.

Composite actions decision: Tool review 1 solves Service Interactions
Interactions that users can have with composites through the tool.

SNS share decision: Tool review 1 solves Composite actions
Users can share composites that they have created on Social Networking services.

Facebook option: Tool review 1 solves SNS share
Share the composite on Facebook.

Twitter option: Tool review 1 solves SNS share
Share the composite on Twitter.

Turn on/off option: Tool review 1 solves Composite actions
Composites that can be executed automatically can be turned on and off.

Rate composite option: Tool review 1 solves Composite actions
Users can give some rating to composites (that they have acquired from other users).

Download composite option: Tool review 1 solves Composite actions
Users can acquire composites that have been created and shared/published by other users.

Edit option: Tool review 1 solves Composite actions
Users can edit composites, either that they have created or that have been created by others.

Delete option: Tool review 1 solves Composite actions
Users can delete composites that they have created.

Rename option: Tool review 1 solves Composite actions
Users can rename composites.

Export as app option: Tool review 1 solves Composite actions
The service composition tool generates new applications for the composite.

Favourite option: Tool review 1 solves Composite actions
Favourite the composite.

Flag option: Tool review 1 solves Composite actions
Flag the composite.

Share/upload/publish option: Tool review 1 solves Composite actions
The tool allows users to share composites with one another.

Copy option: Tool review 1 solves Composite actions
Users can copy composites (either composites of their own, or composites created and published by others).

Execute option: Tool review 1 solves Composite actions
Users can execute composites.

Assign tag to composite option: Tool review 1 solves Composite actions
Users can assign "tags" to composites to classify some aspect of the composite.

View composite execution history option: Tool review 1 solves Composite actions
View the execution history of the composite to show when it has been successfully executed.

Parameters decision: Requirements Gathering solves Composite actions
Parameters of components, such as usernames and passwords.

Set parameters at composition time option: Requirements Gathering solves Parameters
The parameters of the composite are set while the composite is being composed.

Set parameters at runtime option: Requirements Gathering solves Parameters

The parameters of the composite are set after the composite is composed, whilst it is being run.

Set custom icon option: Requirements Gathering solves Composite actions
The user can choose the icon that is presented with the composite.

Review composite option: Requirements Gathering solves Composite actions
The tool allows the user to write reviews for composites.

Save option: Design study solves Composite actions
The user can save composites.

Report spam option: Tool review 2 solves Composite actions
The user can report a component or composite as spam.

Component actions decision: Tool review 1 solves Service Interactions
Interactions that users can have with components through the tool.

Use in composition option: Tool review 1 solves Component actions
Components can be used in composition.

Set parameters option: Tool review 1 solves Component actions
Users can set and edit the parameters of the component.

Activate option: Tool review 1 solves Component actions
Components that require setting up can be activated by the user.

Edit channel option: Tool review 1 solves Component actions
Edit the user account information that has been entered for the channel.

View example option: Tool review 1 solves Component actions
View an example composite within which the component is contained.

Assign tag to component option: Tool review 1 solves Component actions
Add a tag to a component to describe some feature of it

Review component option: Requirements Gathering solves Component actions
Users can review components.

Import component option: Requirements Gathering solves Component actions
Components can be imported from other locations.

Configure components option: Design study solves Component actions
The user can configure the component and change its parameters.

Unlock component option: Tool review 2 solves Component actions
The user can unlock premium components (for example by spending points or money).

E Cognitive walkthrough forms

E.1 Cognitive Walkthrough Form 1: Initial Specification

Date: 1st May 2014 Evaluators: Andy Ridge

Brief Description of the interface

The design tool has a similar interface across all three tasks, with a navigation bar at the top, sidebar containing actions, and a canvas taking up the majority of the screen. A wireframe version of this interface is shown in Figure E-1.

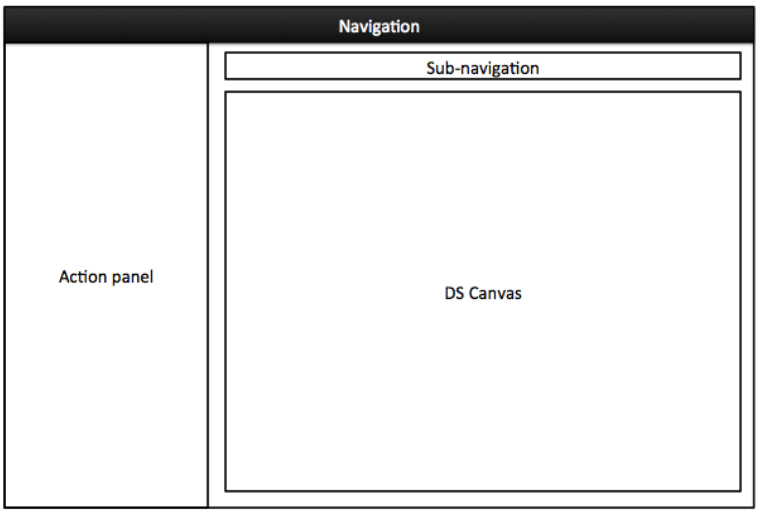


Figure E-1: A wireframe of the general interface of the design space tool.

The different sections of the tool differ in what they present in the action panel, and how the user can interact with the concrete DS hierarchy.

Interface version and status

The interface being evaluated is the final implementation, which is fully-featured and was final as of 30th April 2014.

Suite of User Tasks

The three tasks that will be evaluated are:

1. **Concrete DS creation**: We will be evaluating the task from the point of view of a designer who is creating a concrete DS for a given (generic) domain. Specifically, the designer will be using some other method to generate the design elements that are in the design, and will be using the design tool to record these design elements.

2. **DS profiling**: We will be evaluating the task from the point of view of a designer who is recording a DS profile for a given application in a (generic) domain. Specifically, the designer will be using some other method to identify the design elements in the domain application, and then will be ascribing these design choices to the application within the design tool.
3. **Design creation**: We will be evaluating the task from the point of view of a designer who is creating a design for an application in a (generic) domain. Specifically, the designer will be recording decisions they generate externally, locate decisions within the concrete DS through various means, and edit their design. They will then export this design.

E.2 Cognitive Walkthrough Form 2: User Assumption Form

Assumptions about the user population

We do not assume that the same user is performing each of the tasks that the design tool facilitates, since in a real-world scenario, it is entirely possible that different tasks would be completed by different members of the team. We will discuss the different assumptions about the user, and identify to which task the assumptions apply.

- *Experience in software design* [all]
  The users of the design tool must have basic knowledge about identifying requirements and creating a design for a given piece of software. Note that we don't require expert knowledge in this area. This is most important in the design creation task, although also useful in DS creation and profiling.
  Users would ideally have obtained a proportion of this knowledge through prior experience, although this is not necessary.
- *Knowledge of the domain* [DS creation, DS profiling]
  For DS creation and profiling, knowledge of the domain is useful to make navigation through the concrete DS more efficient.
  In DS profiling, the user could have obtained this knowledge through use of the tool during the DS creation exercise. However for the DS creation stage, users would need to have obtained this knowledge elsewhere – likely through use of EUSC applications.

E.3 Task 1: DS Creation

User Task: Create a concrete DS for a given domain, adding multiple design decisions and options to different sub-spaces within the concrete DS.

Cognitive Walkthrough Forms 3A: Goal Structure Form – Concrete DS Creation

Initial goal structure for creating a new concrete DS:

ALL Create a new, empty concrete DS (On-screen cues & task).
SOME Create a new stage (On-screen cues)
SOME Add new sources (On-screen cues)
ALL Add new design elements to the DS for the first stage of the method (Task & Background knowledge).
SOME Create a new stage (On-screen cues)
SOME Add new sources (On-screen cues)
ALL Add new design elements to the DS for the second stage of the method(Task & Background knowledge).

Cognitive Walkthrough Form 4A: System State Form – DS Creation

Assumptions About System State and User's Environment For normal usage, we assume that another designer has already created a DS that is shown on the home page. Our designer will not be interacting with this DS, but it will be present within the tool. There are no abnormal expectations if the tool were to be used in the real world.

For the DS creation task, we expect the following changes to the system state:

1. New DS creation: The DS does not exist on the home page prior to the task, but following successful completion of the task, the DS is presented on the home page of the design tool.
2. New Stage creation: Assuming that the designer's DS creation method is split into a number of stages, these will be added in the DS creation element of the tool.
3. New source creation: Assuming that some of the design elements that are added to the DS have been identified from a particular source, these need to be added to the tool to be associated with design elements.
4. New design element creation: New design elements are created and stored before they can be linked with other elements in the DS.
5. Concrete DS created: Once the collection of design elements that the designer has identified have been added to and linked with elements in the concrete DS, the tool will contain a standalone object – the concrete DS.

Cognitive Walkthrough Form 5A: Task Evaluation Form – DS Creation

Task considered from the user's viewpoint: The user wants to create a new concrete DS in their chosen domain. They are using some unspecified method to generate the design decisions, potential solutions and links between them that is broken down into stages. They need to record the design elements that they identify, and the sources of these design elements, which are both prior literature and other applications in the domain.

User's initial goals: There is only one likely set of goals for this particular task, although different users may break each of the goals down into more fine-grained goals.

1. Create a new, empty concrete DS (On-screen cues – All users).
2. Create a new stage (On-screen cues – Some users)
3. Add new sources (On-screen cues – Some users)
4. Add new design elements to the DS for the first stage of the method (Task & Background knowledge – All users).
5. Create a new stage (On-screen cues – Some users)
6. Add new sources (On-screen cues – Some users)
7. Add new design elements to the DS for the second stage of the method(Task & Background knowledge – All users).

Designer's action sequence for the task:

For DS creation, there is only one set of actions that can be undertaken for the task to be completed successfully. However, stages can be missed if they are not relevant to the method that is being used by the user of the tool to identify elements of the DS.

1. **Create new concrete DS.** Create a new concrete DS and give it a name.
2. **Edit new concrete DS.** The user will indicate that they want to edit the contents of the new concrete DS that they have created.
3. **Create new stage.** If the method being used to identify the elements of the DS is broken down into stages, then create a stage to represent the initial stage of this method.
4. **Add new source.** If the design element has been identified from a particular source, then add that source.
5. **Add new design element.** Add the first design element to be identified to the DS.
6. **Link design element to relevant root.** Link the newly added design element to the root of the relevant DS.
7. **Repeat for other identified design elements.** Repeat steps 2-5 for all other design elements, linking with the relevant parent element.

Cognitive Walkthrough Form 6A: Action evaluation form – DS Creation

User action: Create new, empty concrete DS **Goal structure:**
Correct goal: Create a new concrete DS.
Mismatch with likely goals: Users may edit a currently existing DS rather than creating a new one.
Choosing the next correct action:
Correct action: The user should click "Create New" at the top of the page, enter the name for their concrete DS, and click "Add".
Knowledge checkpoint: The user should know what they are going to call their DS. System state checkpoint: Pre-existing DSs may cause the design to try to create a new design in an existing DS rather than creating a new concrete DS.
Action availability: Users may miss this action if they see "Create design"
next to an existing DS, or they may not interpret "Create new" as a clickable button.
Action identifiability: Label: "Create new", Location: Top of the page. The location of the identifier is obvious, although the wording is not. It isn't entirely clear what object the user might be creating. The link between the user's goal and the identifier is obvious.
Incorrect action choices: The user may try to create a new design for an existing DS rather than creating a new concrete DS. If users understand their goal then this is unlikely, but may be a problem for novice users.
Executing the action:
Time-outs: No time-out
Hard to do actions: None
System state checkpoint: A new concrete DS is created with the given name.
System response to executed action: The design tool adds the new DS to the list of concrete DSs on the home page, along with links that represent the three actions that the user is able to perform on the DS: "Build/Modify", "Profiling", or "Create design".
Determining progress:
Quit or backup: The user will see that a DS has been created, so will know that they are making progress towards their goal. All users should notice this progress.
Complete and incomplete goals: The user will have completed their initial goal of creating a new, empty concrete DS with the given name. **Modifying the user's current goal structure:**
Formation of new goals: The user will now be able to move onto their next goal of editing the empty concrete DS.

User action: Edit the newly created DS **Goal structure:**
Correct goal: Users will want to add design elements to the concrete DS that they have just created, but will be unable to perform any of the sub-goals without first indicating that they want to edit this DS to the design tool.
Mismatch with likely goals: None
Choosing the next correct action:
Correct action: The user should select "Build/Modify" next to the name of the concrete DS that they have just created.
Action availability: If the user understands their task, then the action is obvious. We expect that few users would miss this action.
Action identifiability: Label" "Build/Modify", Location: Next to the name of the DS that was just created. The identifier is easily linked with the goal of editing the contents of the DS, since none of the other actions that can be performed could be linked with this goal.

Incorrect action choices: None
Executing the action:
Time-outs: None
Hard to do actions: None
System state checkpoint: The user may try to edit the wrong DS, but this is unlikely since they have just created a DS with a given name.
System response to executed action: The DS creation section of the design tool is displayed to the user.
Determining progress:
Quit or backup: Users will be presented with the DS creation section of the tool, so it will be clear that the action has been completed successfully, *Complete and incomplete goals:* The user will have navigated to the DS creation section of the tool and will now be able to edit the contents of the DS to meet their later goals.
Modifying the user's current goal structure:
Formation of new goals: If the user has not already formed the goals of adding stages or sources in their DS creation process, then they should at this point since there are actions available in the DS creation stage that facilitate these goals.

User action: Create new stage. **Goal structure:**
Correct goal: The main goal of the user should be to add a new element to their concrete DS. If applicable, the relevant sub-goal is to create a new stage. If this is not applicable, skip to the next action.
Mismatch with likely goals: Some users will not have this goal structure, so they should skip this action.
Choosing the next correct action:
Correct action: The user should select "Change stage", and use the form to add a new stage.
Knowledge checkpoint: The user needs to know if the method that they are following to create their DS is split into stages or not.
Action availability: It is obvious, the button is prominent on the page, and the form is self-explanatory. Few users would miss the action.
Action identifiability: Label: "Change stage", Location: Top right, above the DS canvas. The identifier is easily linked with both the action and the goal because the same terminology is used.
Incorrect action choices: None
Executing the action:
Time-outs: None
Hard to do actions: None
System state checkpoint: If stages already exist in this DS, then the user may try to change the stage to an already existing stage rather than adding a new one, but this is unlikely because old stages are separated from the section where new stages can be added.
System response to executed action: The current stage is updated to be the stage that the user just added, and the modal dialog where the new stage was added is hidden.
Determining progress:
Quit or backup: The stage that the user added is now shown as the current stage, so it should be clear to the user that their goal has been completed.
Complete and incomplete goals: The user will now be in a position to move onto another goal: either adding or linking elements in the DS, or adding a new data source for new elements.
Modifying the user's current goal structure: *Formation of new goals:* None

User action: Add new source.
Goal structure:
Correct goal: The main goal of the user should be to add a new element to their concrete DS. If applicable, the relevant sub-goal is to create a new source. If this is not applicable, skip to the next section.
Mismatch with likely goals: Some users will not have this goal structure, so they should skip this action. **Choosing the next correct action:**
Correct action: The user should select "Add Source", and use the form to add a new stage.
Knowledge checkpoint: The user needs to know the sources in which the design elements they identified were identified.
Action availability: It is obvious, the button is prominent on the page, and the form is self-explanatory. Few users would miss this action.
Action identifiability: Label: "Add Source", Location: Top left, next to the top corner of the DS canvas. The identifier is easily linked with both the action and the goal because the same terminology is used.
Incorrect action choices: None
Executing the action: *Time-outs:* None
Hard to do actions: None
System response to executed action: The system now has the newly added source, and a status bar is added that indicates the source has been added. The add source dialog is hidden.
Determining progress:
Quit or backup: The dialog is hidden and a status bar is shown indicating that the new source has been added. All users should notice this progress. *Complete and incomplete goals:* The user will now be in a position to move onto another goal: adding or linking elements in the DS.
Modifying the user's current goal structure:
Formation of new goals: None

User action: Add new design element.
Goal structure:
Correct goal: The main goal of the user should be to add a new design element to the DS.
Mismatch with likely goals: None
Choosing the next correct action:
Correct action: The user should select "Add Design Element", and use the form to add a new design element. *Action availability:* It is obvious, the button is prominent on the page, and the form is self-explanatory. Few users would miss this action.
Action identifiability: Label: "Add Design Element", Location: Top left, next to the top corner of the DS canvas, above the button for adding new sources. The identifier is easily linked with both the action and the goal because the same terminology is used.
Incorrect action choices: None
Executing the action:
Time-outs: None
Hard to do actions: None
System response to executed action: The system now has the newly added design element, and a status bar is added that indicates the design element has been added. The add design element dialog is hidden.
Determining progress:
Quit or backup: The user will see that a new design element has been added to the DS, but it may not be clear to them what they should do next, since it is not clear where this design element is now that it has been added.

Complete and incomplete goals: The user will now be able to move on to adding other design elements, or linking the design element in the DS. **Modifying the user's current goal structure:**
Formation of new goals: None

User action: Link design element to relevant root.
Goal structure:
Correct goal: The goal of the user is to link the new element that they have added to the DS with some other element that is already in the DS.
Mismatch with likely goals: None
Choosing the next correct action:
Correct action: The user should click on the element that already exists in the DS, which updates the sidebar to show the information for that design element. The user should then click "Add outgoing link" or "Add incoming link" and then choose the relevant design element from the list of available design elements.
Knowledge checkpoint: The user needs to remember the name of the design element that they added in the previous stage.
Action availability: This action is not obvious to the user, in that it is somewhat counter intuitive that they would need to click on the element in the DS and add the newly added design element to that, rather than being able to select the design element that they added and linking it with an existing design element. A large proportion of users might miss this action.
Action identifiability: Once the user has clicked the design element, the identifier is shown in the sidebar. Label: "Add outgoing link", "Add incoming link", Location: side bar.
Incorrect action choices: None are incorrect, but it is not clear what the correct action is.
Executing the action:
Time-outs: None
Hard to do actions: None
System response to executed action: The representation of the concrete DS is updated to contain the newly linked element.
Determining progress:
Quit or backup: The user will see that their goal has been completed. Few users will not be able to determine that their goal has been completed.
Complete and incomplete goals: The user will have completed their goal of linking their new design element with elements already existing in the space.
assumption form. **Modifying the user's current goal structure:**
Formation of new goals: The user might need to repeat some of the tasks that they've already completed for other new design elements they have identified that need to be added to the concrete DS.

User action: Repeat for other identified design elements.
The final action identified is to repeat the previous stages for all other design elements that have been identified in the user's method. This does not require any further walkthroughs.

E.4 Task 2: DS Profiling

Cognitive Walkthrough From 3B: Goal Structure Form – DS profiling

User Task: For a given concrete DS (created in the previous task), profile an application that has been identified in the domain.

Assumptions About System State and User's Environment For normal usage, we assume that some designer has created a concrete DS that is shown on the home page that the current designer can profile. The concrete DS needs to contain a number of design elements across the different sub-DSs

For the DS profiling task, we expect the following changes to the system state:

1. New DS profile: For each application that the designer chooses to profile, the system now contains the profile linking that application with the concrete DS. These can be exported, as can combinations of these DS profiles.

Cognitive Walkthrough Form 5B: Task Evaluation Form – DS Profiling

Task considered from the user's viewpoint: The user wants to record the design decisions that are considered in applications in the domain, and design solutions that are chosen in those applications. They need to record these decisions and be able to export the results of profiling so that they can be used externally to the tool.

User's initial goals: Initially – when recording the profiling information – there is only one set of goals that the user is likely to have:

1. Add a new application (On-screen cues – All users)
2. Choose a application for which to add profiling information (On-screen cues – All users)
3. Add profiling information (Task – All users)
4. Export profiling information (Task, Background knowledge & On-screen cues – All users)

Different users of the tool might have differing goals for exporting the results of DS profiling:

1. View the profile for a single domain application.
2. Export the profile for a number of domain applications as a heat map from a selected root element.
3. Export the profile for a number of domain applications as a table.

Additional goals:

1. After viewing the profiling information for a single domain application in the design tool, they might want to export it.

Designer's action sequence for the task:

As with the user's goals for DS profiling, recording the profiling information in the concrete DS has a set task sequence. Thus, the actions involving exporting the results of DS profiling are optional:

1. **Create a new application.** Create a new application for which profiling information can be added, and give it a name.
2. **Choose the new application.** Choose the newly added application so that profiling information for it can be added.
3. **Add profiling information for the application.** Indicate that design decisions have been considered in the application being profiled, or that design solutions have been chosen.
4. **Enter Export mode.** Switch the design tool to the mode where profiling information can be exported to be used elsewhere by the user.
5. **Export all profiling information for that application.** Export the profiling information for one of the available applications as a tree.
6. **Export profiling information for all applications as a heat map.** Export the profiling information for all of the available applications as a heat map.
7. **Export profiling information for all applications as a table.** Export the profiling information for all of the available applications as a table.

Cognitive Walkthrough Form 6B: Action evaluation form – DS Profiling

User action: Create a new domain application. **Goal structure:**
Correct goal: The goal of the user is to add a new domain application to the design tool, against which profiling information can be recorded. *Mismatch with likely goals:* None
Choosing the next correct action:
Correct action: The user should click "Add Tool", enter the name of the domain application in the form, and submit it.
Action availability: It is obvious that this action is possible, the button is placed in the action bar below the domain applications that have already been added to the tool.
Action identifiability: Label: "Add Tool", Location: In action bar, below list of domain applications. The link between the identifier and the action is not entirely clear due to a misuse of terminology – domain applications should be referred to as applications, whereas in this case the term 'tool' is used. Any number of users might not interpret this identifier correctly.
Incorrect action choices: No incorrect actions are immediately available, but users may not interpret this as the correct action.
Executing the action:
Time-outs: None
Hard to do actions: None
System response to executed action: The dialog is removed, and the newly added domain application is added to the list of applications in the action bar. **Determining progress:**
Quit or backup: The user will see that their goal has been completed.
Complete and incomplete goals: Having added the domain application, the user will now be able to select it from the list of domain applications and hence start to record profiling information for that application.
Modifying the user's current goal structure:
Formation of new goals: None

User action: Choose the new domain application.
Goal structure:
Correct goal: The user wants to record the profiling information for a domain application, so first they must select that domain application.
Mismatch with likely goals: None
Choosing the next correct action:
Correct action: The user should click on the domain application that they wish to select in the action bar.
Action availability: The action is obvious since the names of the available domain applications are presented next to radio buttons.
Action identifiability: Label: The name of the application, Location: In the action bar, amongst the other domain applications that are available. The link between the identifier and the action is clear since the user knows what application they want to enter the profiling information for.
Incorrect action choices: None
Executing the action:
Time-outs: None
Hard to do actions: None
System response to executed action: The radio button next to the selected application is selected (and all others are de-selected), and the concrete DS is updated to show the profiling

information for the newly selected domain application.
Determining progress:
Quit or backup: The user should see that the currently selected application has been changed, although users may not realise that the concrete DS reflects the profiling information for this application. *Complete and incomplete goals:* Having selected the domain application, users will now be in a position to record profiling information for that domain application.
Modifying the user's current goal structure:
Formation of new goals: None

User action: Add profiling information for the domain application.
Goal structure:
Correct goal: The user wants to record profiling information for a domain application.
Mismatch with likely goals: None
Choosing the next correct action:
Correct action: The user should click the design element in the concrete DS that represents the profiling information they are trying to record. *Action availability:* It is clear that design elements in the concrete DS can be clicked, but it is not obvious what clicking on the design element will do. However, we believe that few users would miss this action since there aren't many actions that can be taken at this point.
Action identifiability: Cursor becomes pointer on mouseover of design element.
Incorrect action choices: The correct action may not be entirely clear, however there aren't many other actions that can be undertaken. **Executing the action:**
Time-outs: None
Hard to do actions: None
System response to executed action: The clicked design element changes colour to indicate that it has been selected as part of the DS profile, and all parent elements in the concrete DS are highlighted accordingly. (A child element cannot be considered without its parents being considered, even if they are considered implicitly.)
Determining progress:
Quit or backup: The design element will be highlighted, which we feel is a clear indication that the interaction has been successful.
Complete and incomplete goals: Having selected this design element, the user will then be able to go on and select all of the other design elements that need to be added to the profile.
Modifying the user's current goal structure:
Formation of new goals: Having chosen a design element (or several), the user might want to export this information.

User action: Enter Export mode.
Goal structure:
Correct goal: The user has recorded profiling information, so they would now want to use or view this information.
Mismatch with likely goals: None
Choosing the next correct action:
Correct action: The user should click "Output profile" in the header bar. *Action availability:* The action is obvious, and prominently placed in the navigation bar. Few users would miss this action. *Action identifiability:* Label: "Output profile", Location: in the navigation bar at the top of the page.
Incorrect action choices: None **Executing the action:**
Time-outs: None

Hard to do actions: None
System response to executed action: The action bar changes to indicate that the user should select the application(s) for which they wish to view the profiling information. The radio buttons next to the applications change to become check boxes, and the "Output profile" button in the navigation bar is selected.
Determining progress:
Quit or backup: The change of the view of the tool will indicate to the user that they have met their goal of switching to output mode. Users may change back to Adding mode if they are unsure, but it is trivial to change back to output mode. Although the terminology across these two modes is inconsistent. *Complete and incomplete goals:* Having changed to output mode, users will now be able to select the applications for which they wish to view or export the profiling information.
Modifying the user's current goal structure:
Formation of new goals: The user might want to view or export the profiling information for one or more of the domain applications in the action bar.

User action: View profiling information for a particular domain application.
Goal structure:
Correct goal: The user would wish to view the profiling information for a given application in the list of available domain applications.
Mismatch with likely goals: A number of different goals are possible at this stage, in that the user might want to view the profiling information for one or more of any of the available applications, they might want to view this information in the design tool, or export it in one of a number of different formats.
Choosing the next correct action:
Correct action: The user should select the check box next to the domain application for which they wish to view the profiling information, and ensure that all check boxes next to other available applications are not selected.
Action availability: If the user understands the purpose of the output mode then this action is obvious.
Action identifiability: The label is the name of the application, which is in the action bar.
Incorrect action choices: The user might select the wrong domain application, or multiple applications, although this is unlikely.
Executing the action:
Time-outs: None
Hard to do actions: None
System response to executed action: The highlights of the design elements in the concrete DS are changed to reflect the profiling information of the selected domain application.
Determining progress:
Quit or backup: It is clear that the user has fulfilled their goal.
Complete and incomplete goals: The goal of viewing the profiling information for the application is now complete. They might optionally move on to one of a number of different other profiling output goals.
Modifying the user's current goal structure:
Formation of new goals: Having seen this profiling information, they might want to export it in one of a number of forms.

User action: Export the profiling information that is shown in the previous action.

Goal structure:
Correct goal: Export the profiling information for the single application that they viewed in the previous action.
Mismatch with likely goals: There is a possible mismatch with any of the other goals that relate to outputting the results of the DS profiling activity.
Choosing the next correct action:
Correct action: The user should select "Graph (.gv)" from the list of possible output formats, and then click "Create Output".
Action availability: It is obvious. Few users would miss this action.
Action identifiability: Label "Create Output", Location: Below the list of domain applications and possible output formats. The create output identifier matches the user's goal, although the output format is not clear.
Incorrect action choices: The user might select the wrong type of output – particularly the other graph-based output option. Although in this case it would not matter.
Executing the action:
Time-outs: None
Hard to do actions: None
System response to executed action: The user is presented with a dialog to allow them to choose where to save the output file. **Determining progress:**
Quit or backup: The user will have downloaded the output file, so it will be clear to them that they have achieved their goal. Note that output graph files require GraphViz[25] to be installed on the user's machine. They may not know that this is the case which could affect their use of the design tool. *Complete and incomplete goals:* The user will have completed their goal of outputting the profiling information for the selected domain application. **Modifying the user's current goal structure:**
Formation of new goals: None

User action: Export profiling information for all applications as a heat map from a selected root element.
Goal structure:
Correct goal: The user wants to export the profiling information for a subset of the DS across several different domain applications, in the form of a heat-map to indicate the relative popularity of the design elements.
Mismatch with likely goals: None
Choosing the next correct action:
Correct action: The user should select the check boxes for the applications for which they wish to export profiling information, click the design element in the concrete DS to be the root of their exported information, select the heat map output format, and then click "Create Output".
Action availability: All of the sub-actions except selecting the root of the output are obvious. The majority of users would be able to export a heat map for the given applications, but some users may miss the selection of the root design element.
Action identifiability: Labels: The application names, "Graph-based heat map (.gv)", "Create Output", Location: in the action bar. There is no identifier for selecting the root, although the tool indicates that a root has been selected afterwards.
Incorrect action choices: The user might export the profiling information for the applications across the whole concrete DS rather than for the section defined by the root node.

[25] `http://www.graphviz.org`

Executing the action:
Time-outs: None
Hard to do actions: None
System response to executed action: The user is presented with a dialog to allow them to choose where to save the output file. **Determining progress:**
Quit or backup: The user will have downloaded the output file, so it will be clear to them that they have achieved their goal. Note that output graph files require GraphViz to be installed on the user's machine. They may not know that this is the case which could affect their use of the design tool.
Complete and incomplete goals: The user will have completed their goal of exporting the profiling information for the chosen applications as a heat map. **Modifying the user's current goal structure:**
Formation of new goals: None

User action: Export profiling information for all applications as a table.
Goal structure:
Correct goal: The user wants to export the profiling information across several different domain applications, in the form of a table to indicate the relative popularity of the design elements.
Mismatch with likely goals: None
Choosing the next correct action:
Correct action: The user should select the check boxes for the applications for which they wish to export profiling information, select the table output format, and click "Create Output".
Action availability: All actions are obvious. Few users would miss this action.
Action identifiability: Labels: The application names, "Table (.csv)", "Create output", Location: in the action bar.
Incorrect action choices: The user might select the wrong applications, or the wrong output format. Although this is unlikely.
Executing the action:
Time-outs: None
Hard to do actions: None
System response to executed action: The user is presented with a dialog to allow them to choose where to save the output file. **Determining progress:**
Quit or backup: The user will have downloaded the output file, so it will be clear to them that they have achieved their goal.
Complete and incomplete goals: The user will have completed their goal of exporting the profiling information for the chosen applications as a table.
Modifying the user's current goal structure:
Formation of new goals: None

E.5 Task 3: Design Generation

Cognitive Walkthrough From 3C: Goal Structure Form – Design generation

User Task: Generate the design for a new domain application using the design tool.

Cognitive Walkthrough Form 4C: System State Form – Design Generation

Assumptions About System State and User's Environment For normal usage, we assume that some designer has created a concrete DS that is shown on the home page that the current designer can use to generate their design. This concrete DS may also have profiling information attached to it for a number of applications.

For the design generation task, we expect the following changes to the system state:

1. New design generated: The user generates a design using the tool, which they later can export.

Cognitive Walkthrough Form 5C: Task Evaluation Form – Design Creation

Task considered from the user's viewpoint: The user wants to generate and record a new design for a application within the chosen domain. This is a very general description of the task, because each designer might have very different processes for generating this design.

User's initial goals: The main goal of the user is to generate a new design for a domain application, and record this design using the design tool. The sub-goals that they have in generating this design will differ wildly, so in this cognitive walkthrough we do not make any assumptions about how the user generates elements of the design, and instead we focus on aspects of navigating the concrete DS, and recording design decisions.

Designer's action sequence for the task Given that the method by which the designer generates the design varies greatly, in this section we will consider all of the disparate actions that the user can undertake:

1. **Show DS element info.** View the information describing a design element within the concrete DS.
2. **Choose a DS element for the design.** Choose a design element in the concrete DS to be included in the design.
3. **Add a custom design solution.** Add a custom design element that is not present in the concrete DS.
4. **Clear design data.** Remove all of the data about the current design from the design tool.
5. **Alter the information being shown.** Change the proportion of design decisions or options that are currently being shown as part of the concrete DS.
6. **Un-choose a design element.** Remove a chosen design element from the design.
7. **Re-choose an un-chosen design element**. Re-select the design element that was removed in the last stage.
8. **Export design.** Export the design that the designer has created.

Cognitive Walkthrough Form 6C: Action evaluation form – Design Creation

User action: Show DS element info.
Goal structure:
Correct goal: The user wants to find out more information about a particular design element within the concrete DS.
Mismatch with likely goals: None
Choosing the next correct action:
Correct action: The user should right click on a design element in the concrete DS representation.
Action availability: It is not immediately obvious that the action is available since there is no indication that the design elements can be right clicked. Furthermore, right click interactions are not normal within Web browsers.
Action identifiability: No identifier
Incorrect action choices: The user might left-click on the design element, which would indicate that they want to consider the design element as part of their design if it is a potential solution, or expand/contract sections of the concrete DS if it is a design decision. If the user left-clicks an option, they would be presented with the information that they needed, but in the context of choosing that decision rather than finding out more about it.
Executing the action:
Time-outs: None
Hard to do actions: None
System response to executed action: A status bar is shown above the concrete DS that shows the name of the design element that has been clicked, and a description of the design element.
Determining progress:
Quit or backup: The status bar indicates that the user has met their goal of finding out more information about the design element. *Complete and incomplete goals:* The user will now be able to decide whether they want to choose this potential design solution. **Modifying the user's current goal structure:**
Formation of new goals: The user might want to choose this decision, meaning they would have to left-click on the element. They should be able to do this from the status bar.

User action: Choose a DS element for the design.
Goal structure:
Correct goal: The user wants to add an element from the concrete DS to their design.
Mismatch with likely goals: Some users may instead try to add a custom goal and give it the same name as an element within the concrete DS.
Choosing the next correct action:
Correct action: The user should click the relevant design solution within the concrete DS, which brings up a dialog. They should then enter the rationale for their design choice and select "Okay".
Action availability: It is not immediately obvious that this action is available, however when the user moves the mouse over the design element, the cursor is changed to a pointer, which is a standard indicator that an element on a Web page can be clicked. Some users may miss this action.
Action identifiability: No identifier.
Incorrect action choices: The user might view information about the design element rather than choosing it, although this is unlikely.
Executing the action: *Time-outs:* None

Hard to do actions: None
System response to executed action: The design element is added to the container that contains all of the chosen design solutions. The design element is highlighted in the concrete DS.
Determining progress:
Quit or backup: The design solution is added to the list of chosen solutions, and is highlights. All users should consider this as progress and hence should not back up.
Complete and incomplete goals: The goal of adding a design element from the concrete DS to their design will have been completed.
Modifying the user's current goal structure:
Formation of new goals: The user might want to remove the design solution that they have just added, if they added it in error or have changed their mind.

User action: Add a custom design solution.
Goal structure:
Correct goal: The user wants to enter their own custom design solution because they have identified an aspect that is not already in the concrete DS. They want to link this solution to one of the design decisions that is already present in the concrete DS.
Mismatch with likely goals: Some users may not have custom solutions that they wish to add, but since we are considering each goal separately this is not applicable.
Choosing the next correct action:
Correct action: The user should enter a name for the custom decision in the "Add custom solution:" text box, and click "Add". This brings up a dialog into which the user can enter a description and rationale for the custom solution, as well as being able to link it with design decisions that are present in the concrete DS.
Action availability: The text box makes it clear what the user should do, but the title of the section is misleading. Misleading terminology may mean that users miss this action.
Action identifiability: Labels: "Other Decision", "Add custom solution", Location: In the side bar, above the results of the design. The problematic terminology means that the link between the identifier and the action is not as strong as it could be.
Incorrect action choices: None
Executing the action:
Time-outs: None
Hard to do actions: None
System response to executed action: After the user has completed the dialog, an element is added to the design underneath the "Custom" heading that contains the name of the custom solution that the user has added.
Determining progress:
Quit or backup: The custom solution is added to the design, so it should be clear that the user is making progress.
Complete and incomplete goals: The user will have completed their goal of adding a custom solution to the design.
Modifying the user's current goal structure:
Formation of new goals: The user might want to remove the custom goal.

User action: Un-choose a chosen design solution.
Goal structure:
Correct goal: The user wants to remove one of the design solutions that they have chosen

for the design.
Mismatch with likely goals: None
Choosing the next correct action:
Correct action: The user should either double click on the chosen design solution, or right click on the design solution and select "Unmake".
Action availability: It is obvious that the user is able to click on the chosen design solution because the cursor changes to a pointer when the user hovers over it. However it isn't clear that they can right click or double click on the design solution. Some users may miss this action.
Action identifiability: There is no label for this identifier.
Incorrect action choices: None – the user might miss the correct action, but it is unlikely that they would carry out an incorrect action.
Executing the action:
Time-outs: None
Hard to do actions: None
System response to executed action: The design element is removed from the list of chosen design solutions, and is added to a hidden list of un-chosen design solutions.
Determining progress:
Quit or backup: The design solution is removed from the list of chosen design solutions, so it should be clear to the user that they are making progress towards their goal.
Complete and incomplete goals: The user has now completed their goal of removing one of their chosen design solutions.
Modifying the user's current goal structure:
Formation of new goals: None

User action: Re-choose an un-chosen design solution.
Goal structure:
Correct goal: If the user has mistakenly removed a design solution from their design, they will want to add it again.
Mismatch with likely goals: None
Choosing the next correct action:
Correct action: The user should show the list of un-chosen design solutions, and then double click on the un-chosen design solution that they wish to re-choose. They may also right click on the un-chosen design solution and click "Choose".
Action availability: It is not obvious that this is the correct action because by default the list of un-chosen design solutions is hidden.
Action identifiability: Label: "Un-Chosen Options", the name of the design solution, and "Choose", Location: in the action bar, and in the information dialog about the design solution.
Incorrect action choices: None – the user might miss the correct action, but it is unlikely that they would carry out an incorrect action.
Executing the action:
Time-outs: None
Hard to do actions: None
System response to executed action: The design solution is re-added to the list of chosen design solutions. It is removed from the list of un-chosen design solutions.
Determining progress:
Quit or backup: The design solution is added to the list of chosen design solutions, and removed from the list of un-chosen solutions, so it should be clear to the user that they are making progress towards their goal.

Complete and incomplete goals: The user has now completed their goal of re-adding a removed design solution.
Modifying the user's current goal structure:
Formation of new goals: None

User action: Clear design data.
Goal structure:
Correct goal: The user wants to clear all design data and start again.
Mismatch with likely goals: None
Choosing the next correct action:
Correct action: Click "Clear Data"
Action availability: The action is obvious, users are very unlikely to miss this action. Although they may do it by mistake.
Action identifiability: Label: "Clear Data", Location: At the top of the page in the navigation bar. The link between the identifier and the action is fairly clear, although it would be possible for the user to misconstrue what data is to be cleared. Thus, the terminology in the identifier could be clearer.
Incorrect action choices: None
Executing the action:
Time-outs: None
Hard to do actions: None
System response to executed action: The design data is cleared from the tool.
Determining progress:
Quit or backup: All of the information on the design is removed from the tool. The user cannot quit or backup, which could be an issue if done mistakenly.
Complete and incomplete goals: The user will have completed their goal of clearing all of the data for the current design.
Modifying the user's current goal structure:
Formation of new goals: The user might want to undo the clearing of the data, which is currently impossible.

User action: Alter the information being shown.
Goal structure:
Correct goal: The user is being overloaded by the information being presented within the concrete DS, they wish to reduce the information being displayed to them.
Mismatch with likely goals: None
Choosing the next correct action:
Correct action: The user should drag either the decision slider or the solution slider to a value lower than 100%. *Action availability:* It is obvious, not many users would miss this action.
Action identifiability: Label: "Decisions showing", "Solutions showing", Location: At the top of the action bar. The identifiers reflect the action because they present the current percentage of decisions or solutions that are being displayed, which changes as the user moves the slider.
Incorrect action choices: None
Executing the action:
Time-outs: None
Hard to do actions: None
System response to executed action: The amount of information in the concrete DS is reduced.

The slider moves following the user's mouse, and the percentage label changes depending on the value of the slider.
Determining progress:
Quit or backup: The user will see from the label, slider, and contents of the concrete DS that the amount of information has changed. If they wish to revert this, they simply have to move the slider back to its original position.
Complete and incomplete goals: The user's goal of changing how much information is being displayed in the concrete DS is completed.
Modifying the user's current goal structure:
Formation of new goals: The user might want to choose other elements in the DS based on the reduced amount of data that it presents.

User action: Export design.
Goal structure:
Correct goal: The user has finished generating their design, and now wants to export it.
Mismatch with likely goals: None
Choosing the next correct action:
Correct action: The user should select "Export Design"
Action availability: It is obvious, the terminology is clear and the button is prominently placed.
Action identifiability: Label: "Export Design", Location: In the navigation bar at the top of the page. The link between the identifier and the action is clear.
Incorrect action choices: None
Executing the action:
Time-outs: None
Hard to do actions: None
System response to executed action: The user is presented with a dialog where they are able to choose where to save the output of the design.
Determining progress:
Quit or backup: The user is able to download the output of the process, so it should be clear to them that they have completed their goal.
Complete and incomplete goals: The user will have completed their goal of exporting the design.
Modifying the user's current goal structure:
Formation of new goals: None

E.6 Cognitive Walkthrough 7: Walkthrough summary form

Since our tool is specialised for designers, we were able to make a number of specific assumptions about the users of our tool, which were reasonable in the context of all three of the tasks that we assessed in this cognitive walkthrough. Summaries for the findings of each of the walkthroughs are provided below.

Summary: DS Creation

In the DS creation phase, we were able to match the user's goals with the designer's action sequence very closely. However, a number of problems were found within these actions. First, considering the home page of the design tool:

- The terminology on the home page is not clear, or consistent across the different actions that can be performed there. A review of the terminology that is used here is needed to ensure clarity.
- The location of the action to create a new concrete DS needs to be moved into a more logical position amongst the list of concrete DSs, rather than being on its own.
- The user may create a concrete DS with a name the clashes with a current DS, which may cause confusion. The system should check for clashes in DS names. Users should also be able to rename existing DSs.

Next, we consider the issues that were identified in the section of the tool where design elements can be added to the concrete DS:

- Auto-complete text boxes that are used within design element linking and source lookup are initially empty and don't show any hints until the user enters something, which is confusing. These need to show all information by default.
- It isn't clear what design elements have been added to the DS but not linked to anything. Once a design element has been added, it should be shown in a list of design elements that have been added to the DS but not linked with any other design elements already in the DS.
- Users should be able to link new design elements with old ones, not just link old ones with new ones.

Summary: DS Profiling

In the DS profiling phase, the initial set of goals matched with the designer's projected actions, however it was unclear what proportion of users would have the goals relating to exporting the results, so these did not match. We found the following problems with this section of the tool:

- The terminology used within this section of the tool is inconsistent – 'tool' should always refer to the design tool, and application should always refer to the domain applications that are being profiled. The terminology relating to entering profiling data and exporting the results of profiling is also inconsistent.
- It isn't always clear what information is being shown in the concrete DS, a label is required to make this explicit.
- It isn't clear what action will occur when the user clicks on an element in the concrete DS, some information is required to inform the user of this.

- The implications of choosing the different output formats are not clear, some information is required to inform the user of what each format means to the user is required.
- Some of the output formats require the user to have installed GraphViz, this should be indicated before they download the exported file.
- It isn't clear how to choose the root node of the data to be exported, or indeed what it means to choose this root. This needs to be explained better.

Summary: Design Generation

The design generation section has matches between the disparate goals that a user might have and actions that the designer of the tool has identified. However, we don't consider these together because of the different approaches that users might take in generating a design. The problems identified in this section are:

- It isn't clear that clicking on an element in the DS chooses them, some prompt is required to indicate that this is the case.
- It isn't clear that the user can right-click on an element in the DS, or what right-clicking on any of these elements will do.
- The user needs to be able to act on the information that they are provided with when they view the information of a design element. A "choose" button should be added to the status bar.
- It isn't clear what the box of chosen options is, better labelling is required.
- The terminology in the action bar is inconsistent. As with the other sections of the design tool, a review of the terminology is required.
- Clearing design data needs an "Are you sure?" to make it more difficult for users to clear the design data by mistake.
- Clear data needs to be "Clear Design Data" to ensure that it is clear to the user as to what data will be cleared from the tool.
- Rather than clearing the data entirely, the cleared data should be moved somewhere where it is not immediately accessible, but can be retrieved later if necessary.

F Design Study Materials

F.1 Consent Form

Study Overview

This study aims to assess how assistance can be provided in creating the conceptual design for a Service Composition tool/application. To do this, we will be asking you to create a conceptual design for a Service Composition application, given a particular type of help. The session should take no more than an hour.

During the session, you will be introduced to Service Composition, as well as being shown what we want you to create [15 minutes]. We'll also present you with a questionnaire to see what experience you've had with Service Composition (or similar concepts) before [5 minutes]. The main stage of the session will be to create a conceptual design for a Service Composition tool/application [max 30 minutes]. Finally we'll give you another questionnaire to ask how you think the session went [10 minutes]. The session will be recorded on video and then the audio from the session will be transcribed anonymously in order to find any problems that you had during the session. During this process, the data will be stored securely.

Important Information

- All data collected during this study will be recorded such that your individual results are anonymous and cannot be traced back to you.
- Your results will not be passed to any third party and are not being collected for commercial reasons.
- Participation in this study does not involve physical or mental risks outside of those encountered in everyday life.
- All procedures and information can be taken at face value and no deception is involved.
- You have the right to withdraw from the study at any time and to have any data about you destroyed. If you do decide to withdraw, please inform the experimenter.

By signing this form you acknowledge that you have read the information given above and understand the terms and conditions of this study.

Name:			
Age:		Sex:	
Occupation			
Signed:		Date:	

Experimenter: Andy Ridge, Dept. of Computer Science. A.D.Ridge@bath.ac.uk

F.2 Pre-Study Questionnaire

Service Composition

1. Have you ever used a Service composition tool/application as part of your day-to-day routine – either at work or for pleasure? (Please choose *one* answer)

. Yes ☑
. No ☑

2. If so, please list the Service Composition tools or applications that you've used:

Design & Software

For the purposes of this questionnaire, we consider a piece of software to be an application for which you have received or conceived of a specification, and coded a solution that meets this solution.

1. Have you designed software before?

. Yes ☑
. No ☑

2. On what platforms have you designed software?

. Desktop ☑
. Mobile ☑
. Tablet ☑
. Web ☑
1. Other:

3. For what purposes have you designed software?

. Commercial ☑
. Academic ☑
. Personal project ☑
1. Other:

4. Roughly how many pieces of software have you designed?

5. What programming languages do you know?

6. Have you ever 'mashed-up' two or more APIs in a single application? (Please choose *one* answer)

. Yes ☑
. No ☑

7. Do you design software as part of your current role or job? (Please choose *one* answer)

. Yes ☑
. No ☑

F.3 General Demographics Questionnaire

1. How old are you?

2. What is your sex? (Please circle *one* answer)

. Male
. Female

3. What is the highest degree or level of education you have completed? If currently enrolled please indicate the highest you have attained previously. (Please circle *one* answer)

. None
. GCSEs or equivalent
. A/AS or equivalent
. BSc/BA or equivalent
. MSC/MA or equivalent
. PhD or equivalent

4. In what field is your highest qualification?

5. What is your current employment status?

. Unemployed
. Self-employed
. Employed
. Student
. Retired
. Unable to work

F.4 Introduction to Session/Task

The task that you will need to complete is to create a conceptual design for a Service Composition tool/application.

A conceptual design is the first part of the design phase, where the aim is simply to list the ideas that make up the features of the application that is being designed. This is done before explicitly thinking about any specific technologies, architectures, or interface designs, although each of these can be affected by the conceptual decisions made earlier on.

We'll be providing you with some assistance in finding and making these decisions, which will be described later.

For each concept, we want you to record the following:

1. A name for the concept (max 5 words).
2. A description for the concept (as long as you want).
3. The reason that you made this decision (this will be called the rationale).

Example Conceptual Design: Video Chat Application

To help you visualise what we want you to create, we've made an example for something we hope you're more familiar with: a video chat application. We'll only show some of the decisions here to save space, but they should hopefully give you an idea of what we want you to produce:

Name:	**Contact list**
Description:	The application shows the user a list of their contacts.
Rationale:	*Users need a list of contacts so that they can keep track of the people they chat with easily.*

Functional Make video call
The application should allow users to make video calls with other users of the application.
This is a fundamental feature of a video chat application.

Receive video call
The application should allow the user to receive video calls from their contacts.
Video calls need to be two-way.

Text chat if limited bandwidth
The application should allow users to communicate with each other by text only if there isn't sufficient bandwidth for them to use video.
Users should be able to talk to each other even if they have limited bandwidth, for instance on a mobile network.

Group chat
Users should be able to video- or text-chat with multiple users at once.
Users may want to chat with multiple friends at once, and rather than having to open multiple

instances of chats with single friends, they should be able to chat with them all at once.

Non-functional All users should be accounted for
The application should allow users of various different skill levels and backgrounds to use the application.
Video chat applications could be used by anyone – even children – so the application must be supportive of lots of different types of user

Help
The application should provide help systems that users can use to find out more information.
There may be tasks that the user wants to perform with the application that they are unsure how to perform, and help systems can help them to find out how to do so.

Structural The application should be multi-platform
Users should be able to use the application on various platforms as well as being able to communicate with users on other platforms.
The application is more useful because the user can communicate with their friends regardless of their preferences of mobile/desktop platform.

Peer-to-peer communication
The application should allow users' devices to connect directly to one another rather than through a centralised server.
If the application was centralised, then if the server goes down, users won't be able to talk to one another, whereas a p2p system allows direct communication between users.

Entities Add new contact
The application allows users to add new contacts to their contact list.
Users need a way of adding new people to their list of contacts.

Video Resolution or quality
The application should allow the users to see the quality of the video stream.
Users should be able to see if the quality drops below a certain level.

Contact name
Contacts' names should be shown to the user.
Users need to be able to see who they can contact with

Contact availability
The availability of a contact shows whether the user is online, available, busy, offline, etc.
Users need to know if their contacts are available to chat before initiating a video call.

F.5 Introduction to Service Composition

Service Composition (SC) is the process of combining a series of "components" together to create some new functionality or application. These components can be anything from software to look up the weather, make social network updates, to web-controlled light bulbs.

Service composition is a way of allowing you to connect the services you use together without the developers of that service making the connections manually.

Broadly speaking, there are 4 main stages in the SC process that need to be accounted for:

1. Discovery of components.
2. Composition of components to create a composite.
3. Verifying the created composite works, and is what was required.
4. Distributing and executing the created composite.

EUSC applications are available on different contexts: mobile, web and desktop. They are also available across several domains, including mobile phone-based interactions, social media, multimedia, etc. There are also different levels of complexity in the applications, which is based on the number of components that can be used in each composition, or how these can be combined together – they might be linearly connected, or they could branch or loop, etc.

We'll now be showing you 7 different EUSC applications, and performing different tasks on each:

1. **Tasker**: Tasker is an Android-based Service Composition application that allows users to access features of the phone and react to changes across those features. You can also change things like settings.
   Task: Launch a music app when headphones are plugged in (1)
2. **Atooma:** Atooma is an Android-based Service Composition application that allows users to connect components together. Atooma allows users to compose Web-based services (e.g. Facebook, Google Drive), as well as mobile phone specific services such as SMSs and the accelerometer in the phone. There isn't as much control over settings as there is in Tasker.
   Task: If receive SMS while driving, read it out loud (2)
3. **AutomateIt**: AutomateIt is similar to Tasker and Atooma, in that it is an Android app that provides access to many phone-related services, although fewer than either Tasker or Atooma.
   Task: Turn on WiFi when the phone is plugged in (3)
4. **IFTTT**: IFTTT is currently available both on Web and iOS, and allows users to connect services together, based on the phrase: If [this] then [that]. Where [this] and [that] are the services that need to be connected together. The web-based version operates on mostly web services (Facebook, weather lookup, Evernote, etc.), with some in-home services also available, like motion sensing or web-controlled light bulbs.
   Task: Text me the weather every morning (4)
5. **Yahoo! Pipes**: Yahoo! Pipes allows the user to combine RSS feeds to make custom list of news articles. The services that Yahoo! Pipes provides allow users to manipulate RSS feeds to let the user do things like filter or combine these feeds.
   Task: Add your name to all of the news items on the BBC news feed (5)
6. **Quartz Composer**: Quartz Composer is a multimedia Service Composition application that allows you to compose various visual effects in order to create compound

visual effects such as image manipulations or screensaver-like visualisations.
Task: Make a series of shapes follow the mouse around the screen (6)

7. **Automator**: Automator is an OSX-based desktop application that is meant to automate tasks that you might want to complete using the applications in OSX.
Task: Get all images from a Wikipedia page (7)

F.6 Tasks

Tasker – Launch a music app when the headphones are plugged in (1):

1. Profiles → New profile → State - ¿hardware → headset plugged
2. No mic → back
3. New task → enter "Launch mp3"
4. Add action → Alert → Menu → enter "Mp3"
5. Add item → Action → app → Load app → "Amazon MP3" → back
6. Add action → Action → app → Load app → "Play music" → back
7. Back → exit Tasker → plug in headphones

Atooma – If receive SMS while driving, read it out loud (2):

1. New Atooma → Mobile → Messages → SMS Received → Next → OK
2. + IF → Mobile → GPS → Speed) → More Than → 20mph →
3. DO → Mobile → Text to speech → Speak → "Messages ¿¿ SMS received ¿¿ Message" → OK
4. = → Enter name → Save

AutomateIt – Turn on WiFi when the phone is plugged in (3):

1. Add rule → Power connected trigger
2. Set WiFi State action → WiFi enabled → next
3. Enter name → Save

IFTTT – Text me the weather every morning (4):

1. Create a recipe → click "this" → Weather → Today's weather report → Set time to "07:00" → Create Trigger
2. Click "that" → SMS → Send me an SMS → Customise message → Create Action
3. Enter description → Create recipe

Yahoo! Pipes – Add your name to all of the news items on the BBC news feed (5):

1. Fetch feed → Enter URL: "http://feeds.bbci.co.uk/news/rss.xml"
2. Filter → Block, all, item.description, contains, Apple
3. Loop → String builder inside
4. Text → name: text, prompt: text, Default: [Your name]
5. Wire text output to String Builder text
6. Loop assign results to item.author
7. Loop → String builder inside → String "item.title" → String "[" → String "item.author" → String "]" → Assign results to item.title
8. Wire Filter to first Loop → Wire first Loop to second Loop
9. Wire second Loop to Pipe Output

Quartz Composer – Make a series of shapes follow the mouse around the screen (6):

1. Add "Clear"
2. Add "Particle System" → open parameters pane → set Colour
3. Add "Lenticular halo" → Connect "Image" of Lenticular Halo to "Image" of Particle System
4. Add "Mouse" → Connect "X Position" of Mouse to "X Position" of Particle System → Connect "Y Position" of Mouse to "Y Position" of Particle System.

Automator – Get all images from a Wikipedia page (7):

1. Create new ““Application””
2. Add “Get current Webpage from Safari”
3. Add “Get Image URLs from Webpage” → Get URLs of images “on these webpages”
4. Add “Download URLs” → Where ““Downloads””
5. Add “Import Files into iPhoto” → “New album”, Enter “Automator” → Tick “Delete the source images after importing”

F.7 Participant EUSC Application Order

Table F.1: The order in which participants were presented with EUSC applications in the design space study (Participants 1-20).

1	1	IFTTT	Automator	Quartz Composer	Yahoo! Pipes	Atooma	Tasker	AutomateIt
		Automator	Tasker	IFTTT	Atooma	AutomateIt	Yahoo! Pipes	Quartz Composer
2	2	IFTTT	Atooma	Quartz Composer	Tasker	Automator	Yahoo! Pipes	AutomateIt
		Yahoo! Pipes	Tasker	Quartz Composer	Atooma	IFTTT	AutomateIt	Automator
3	3	Quartz Composer	Yahoo! Pipes	Tasker	AutomateIt	Automator	IFTTT	Atooma
		Tasker	Automator	Quartz Composer	Yahoo! Pipes	AutomateIt	IFTTT	Atooma
4	2	IFTTT	Automator	Tasker	Atooma	AutomateIt	Yahoo! Pipes	Quartz Composer
		Quartz Composer	Atooma	AutomateIt	Tasker	Automator	IFTTT	Yahoo! Pipes
5	4	Quartz Composer	IFTTT	Automator	AutomateIt	Atooma	Yahoo! Pipes	Tasker
		Atooma	IFTTT	Automator	Tasker	Yahoo! Pipes	AutomateIt	Quartz Composer
6	1	IFTTT	Quartz Composer	Automator	Tasker	AutomateIt	Atooma	Yahoo! Pipes
		Quartz Composer	Tasker	AutomateIt	Yahoo! Pipes	Atooma	IFTTT	Automator
7	4	IFTTT	Atooma	Quartz Composer	Yahoo! Pipes	Tasker	Automator	AutomateIt
		IFTTT	Quartz Composer	Atooma	Tasker	AutomateIt	Yahoo! Pipes	Automator
8	3	Automator	AutomateIt	Tasker	Yahoo! Pipes	Quartz Composer	Atooma	IFTTT
		Automator	Yahoo! Pipes	Tasker	IFTTT	AutomateIt	Quartz Composer	Atooma
9	5	Quartz Composer	Yahoo! Pipes	Atooma	Automator	IFTTT	Tasker	AutomateIt
		IFTTT	Atooma	AutomateIt	Quartz Composer	Automator	Yahoo! Pipes	Tasker
10	5	IFTTT	Yahoo! Pipes	Tasker	Atooma	AutomateIt	Automator	Quartz Composer
		Quartz Composer	Yahoo! Pipes	AutomateIt	Atooma	Automator	IFTTT	Tasker
11	4	Quartz Composer	Atooma	Yahoo! Pipes	Automator	AutomateIt	Tasker	IFTTT
		Quartz Composer	Automator	AutomateIt	Yahoo! Pipes	IFTTT	Atooma	Tasker
12	5	Automator	Atooma	Tasker	AutomateIt	Yahoo! Pipes	Quartz Composer	IFTTT
		Automator	Atooma	Tasker	Quartz Composer	IFTTT	Yahoo! Pipes	AutomateIt
13	1	Yahoo! Pipes	IFTTT	Quartz Composer	Tasker	AutomateIt	Atooma	Automator
		IFTTT	Atooma	Automator	Tasker	AutomateIt	Quartz Composer	Yahoo! Pipes
14	2	Tasker	IFTTT	AutomateIt	Atooma	Automator	Quartz Composer	Yahoo! Pipes
		Tasker	Automator	Quartz Composer	Atooma	Yahoo! Pipes	AutomateIt	IFTTT
15	3	Quartz Composer	Automator	Yahoo! Pipes	Tasker	IFTTT	AutomateIt	Atooma
		IFTTT	Tasker	Quartz Composer	AutomateIt	Yahoo! Pipes	Atooma	Automator
16	1	Yahoo! Pipes	IFTTT	Quartz Composer	AutomateIt	Atooma	Tasker	Automator
		Atooma	AutomateIt	Yahoo! Pipes	IFTTT	Automator	Quartz Composer	Tasker
17	2	Automator	IFTTT	Tasker	AutomateIt	Atooma	Yahoo! Pipes	Quartz Composer
		Yahoo! Pipes	Quartz Composer	Atooma	Tasker	Automator	AutomateIt	IFTTT
18	3	Quartz Composer	Yahoo! Pipes	Tasker	IFTTT	Atooma	AutomateIt	Automator
		Tasker	IFTTT	Automator	AutomateIt	Yahoo! Pipes	Quartz Composer	Atooma
19	5	AutomateIt	Tasker	Atooma	Automator	Yahoo! Pipes	IFTTT	Quartz Composer
		IFTTT	Tasker	Quartz Composer	Atooma	Yahoo! Pipes	AutomateIt	Automator
20	4	Automator	IFTTT	Atooma	Tasker	Quartz Composer	AutomateIt	Yahoo! Pipes
		Yahoo! Pipes	IFTTT	Tasker	Automator	Quartz Composer	AutomateIt	Atooma

Table F.2: The order in which participants were presented with EUSC applications in the design space study (Participants 21-40).

21	3	Automator	IFTTT	Yahoo! Pipes	Tasker	Quartz Composer	AutomateIt	Atooma
		AutomateIt	Automator	Atooma	Quartz Composer	IFTTT	Yahoo! Pipes	Tasker
22	2	Yahoo! Pipes	Atooma	Quartz Composer	Automator	IFTTT	AutomateIt	Tasker
		Yahoo! Pipes	AutomateIt	Quartz Composer	IFTTT	Tasker	Automator	Atooma
23	5	IFTTT	AutomateIt	Quartz Composer	Atooma	Tasker	Yahoo! Pipes	Automator
		Tasker	Yahoo! Pipes	Atooma	Quartz Composer	IFTTT	AutomateIt	Automator
24	1	Tasker	Quartz Composer	IFTTT	Automator	Atooma	AutomateIt	Yahoo! Pipes
		Yahoo! Pipes	Tasker	IFTTT	Atooma	Automator	AutomateIt	Quartz Composer
25	4	Quartz Composer	Tasker	IFTTT	Automator	AutomateIt	Yahoo! Pipes	Atooma
		Atooma	IFTTT	Quartz Composer	Automator	Yahoo! Pipes	AutomateIt	Tasker
26	2	Automator	Quartz Composer	Atooma	Tasker	IFTTT	Yahoo! Pipes	AutomateIt
		AutomateIt	Yahoo! Pipes	Atooma	Tasker	IFTTT	Automator	Quartz Composer
27	4	Yahoo! Pipes	Quartz Composer	IFTTT	Atooma	AutomateIt	Automator	Tasker
		Atooma	AutomateIt	Quartz Composer	Tasker	IFTTT	Automator	Yahoo! Pipes
28	3	AutomateIt	Automator	Yahoo! Pipes	Quartz Composer	IFTTT	Tasker	Atooma
		Tasker	Quartz Composer	Yahoo! Pipes	IFTTT	Automator	AutomateIt	Atooma
29	5	Yahoo! Pipes	Tasker	Quartz Composer	AutomateIt	IFTTT	Atooma	Automator
		Automator	Atooma	Quartz Composer	Yahoo! Pipes	Tasker	AutomateIt	IFTTT
30	1	Yahoo! Pipes	Automator	AutomateIt	Quartz Composer	Atooma	IFTTT	Tasker
		Automator	Atooma	Yahoo! Pipes	IFTTT	AutomateIt	Quartz Composer	Tasker
31	2	IFTTT	Quartz Composer	Yahoo! Pipes	Automator	Tasker	Atooma	AutomateIt
		Automator	Tasker	Atooma	IFTTT	Yahoo! Pipes	Quartz Composer	AutomateIt
32	5	AutomateIt	Atooma	Yahoo! Pipes	Quartz Composer	IFTTT	Tasker	Automator
		IFTTT	Automator	AutomateIt	Quartz Composer	Atooma	Tasker	Yahoo! Pipes
33	4	Tasker	Atooma	AutomateIt	Automator	Yahoo! Pipes	Quartz Composer	IFTTT
		Atooma	Yahoo! Pipes	Automator	AutomateIt	Quartz Composer	Tasker	IFTTT
34	1	Automator	IFTTT	Yahoo! Pipes	Atooma	Quartz Composer	Tasker	AutomateIt
		Automator	IFTTT	Tasker	Atooma	AutomateIt	Quartz Composer	Yahoo! Pipes
35	3	Automator	Quartz Composer	AutomateIt	Atooma	IFTTT	Yahoo! Pipes	Tasker
		Quartz Composer	Yahoo! Pipes	IFTTT	Tasker	Automator	Atooma	AutomateIt
36	5	Atooma	Yahoo! Pipes	Quartz Composer	Automator	AutomateIt	IFTTT	Tasker
		Quartz Composer	Tasker	AutomateIt	Yahoo! Pipes	Automator	IFTTT	Atooma
37	2	Automator	Tasker	AutomateIt	Yahoo! Pipes	Atooma	IFTTT	Quartz Composer
		Yahoo! Pipes	IFTTT	Atooma	AutomateIt	Quartz Composer	Tasker	Automator
38	1	Atooma	Quartz Composer	Automator	IFTTT	Tasker	Yahoo! Pipes	AutomateIt
		Automator	Yahoo! Pipes	Quartz Composer	AutomateIt	IFTTT	Tasker	Atooma
39	3	Yahoo! Pipes	Quartz Composer	Atooma	Tasker	IFTTT	AutomateIt	Automator
		Atooma	Yahoo! Pipes	Automator	Quartz Composer	Tasker	AutomateIt	IFTTT
40	4	IFTTT	AutomateIt	Atooma	Quartz Composer	Automator	Yahoo! Pipes	Tasker
		AutomateIt	Tasker	Automator	IFTTT	Atooma	Yahoo! Pipes	Quartz Composer

F.8 Instructions to the Task and Tool

Instructions: Condition 1

This is the part of the study where you'll be asked to create the conceptual design for a SC application. We want you to list the features

We want you to make a list of the concepts that you would pick for your own application for performing SC. There are no right and wrong answers in this process. If you have any questions or are stuck at any point then please ask. This stage will be limited to 60 minutes, but you can indicate you want to finish at any point before that.

In this session, we'll make as many of the SC applications available to you as you wish. We introduced you to 4 earlier, but there are also several others. The list below shows you what they are and where to find them:

1. OSX [Desktop shortcuts]: Automator, Quartz Composer
2. Web browser [Open browser tabs]: Yahoo! Pipes, IFTTT
3. Android [Home screen shortcuts]: Tasker, AutomateIt, Atooma

You can use any of these applications as much as you like throughout the session.

You should also feel free to ask any questions that you want throughout the session.

We will also present you with a video to demonstrate how to use the application.

Design Tool Controls Reference *Right click on a choice you've made:*
View properties and actions for that choice.

Double click on a choice you've made:
Remove that choice

Instructions: Conditions 2 & 3

In this session, we'll provide you with a Web-based representation of part of what we call a design space model for Service Composition applications. The model that you'll be using shows a collection of the decisions that you might want to consider when coming up with the design of an SC application. Although you should note that it is impossible for this to present all of the decisions that could be made.

We want you to make a list of the concepts that you would pick for your own application for performing SC. There are no right and wrong answers in this process. If you have any questions or are stuck at any point then please ask. This stage will be limited to 60 minutes, but you can indicate you want to finish at any point before that.

The decisions that are presented in the model are split into 4 different categories:

1. **Functional**: The functions that the SC application needs to perform.
2. **Non-functional**: Other aspects of the SC application such as its representation and how the user can interact with it.
3. **Structural**: Aspects of the structure or architecture of the SC application, or how the services are connected to one another.
4. **Entity**: Decisions that relate to the services that are composed using the application.

The model is built into a Web page that allows you to add the concepts that make up the design that you're going to make. You can browse around the categories of decision, which are linked together into a hierarchy. If you want to show and hide parts of the hierarchy, you can click on decisions and their children in the hierarchy will be shown or hidden.

You should consider that the decisions in the model aren't the only decisions that need to be made.

As well as this model of decisions, we will provide you with access to the applications that you were introduced to in the introduction, both from the applications themselves and the videos of the tasks. The list below shows you what they are and where to find them:

1. OSX [Desktop shortcuts]: Automator, Quartz Composer
2. Web browser [Open browser tabs]: Yahoo! Pipes, IFTTT
3. Android [Home screen shortcuts]: Tasker, AutomateIt, Atooma

You can use any of these applications as much as you like throughout the session.

You should also feel free to ask any questions that you want throughout the session.

We will also present you with a video to demonstrate how to use the application.

Design Tool Controls Reference *Right click on a choice you've made:*
View properties and actions for that choice.

Double click on a choice you've made:
Remove that choice.

Right click on a design decision:
View information about that design decision.

Left click on a decision:
Show or hide part of the tree underneath that decision.

Instructions: Conditions 4 & 5

In this session, we'll provide you with a Web-based representation of what we call a design space model for Service Composition applications. This design space model is a collection of decisions that can be made in the design of an SC application, as well as various options that can help to solve that decision. These are the decisions that you might want to consider when designing an SC application. Although you should note that it is impossible for this to present all of the decisions that could be made.

We want you to make a list of the concepts that you would pick for your own application for performing SC. There are no right and wrong answers in this process. If you have any questions or are stuck at any point then please ask. This stage will be limited to 60 minutes, but you can indicate you want to finish at any point before that.

The decisions and potential solutions that are presented in the model are split into 4 different categories:

1. **Functional**: The functions that the SC application needs to perform.
2. **Non-functional**: Other aspects of the SC application such as its representation and how the user can interact with it.
3. **Structural**: Aspects of the structure or architecture of the SC application, or how the services are connected to one another.
4. **Entity**: Decisions that relate to the services that are composed using the application.

The model is built into a Web page that allows you to add the concepts that make up the design that you're going to make. You can browse around the categories of decision, which are linked together into a hierarchy. If you want to choose options in the model, you can double click on them and you will be asked to enter a reason for choosing that option. You can also show or hide parts of the hierarchy by clicking on a decision to hide the sub-hierarchy below it.

You should also consider that these are not all the decisions needed for a Service Composition tool. You are also able to choose more than one option for each decision in the model.

As well as this model of decisions, we will provide you with access to the tools that you were introduced to in the introduction, both from the tools themselves and the videos of the tasks. The list below shows you what they are and where to find them:

1. OSX [Desktop shortcuts]: Automator, Quartz Composer
2. Web browser [Open browser tabs]: Yahoo! Pipes, IFTTT
3. Android [Home screen shortcuts]: Tasker, AutomateIt, Atooma

You can use any of these tools as much as you like throughout the session.

You should also feel free to ask any questions that you want throughout the session.

We will also present you with a video to demonstrate how to use the tool.

Design Tool Controls Reference *Right click on a decision or potential option:*
View properties and actions for that decision or option.

Left click on a decision:
Show or hide part of the tree underneath that decision.

Left click on an option:
Choose that option.

Right click on a choice you've made:
View properties and options for that choice.

Double click on a choice you've made:
Remove that choice.

F.9 NASA TLX

We are not only interested in assessing the design output of the task, but also the experiences you had during the task. In other words, we are examining the "workload" you experienced. Workload is a difficult concept to define precisely, but a simple one to understand generally. The factors that influence your experience of workload may come from the task itself, your feelings about your own performance, how much effort you put in, or the stress and frustration you felt. The workload contributed by different task elements may change as you get more familiar with a task, perform easier or harder versions of it, or move from one task to another. Physical components of workload are relatively easy to conceptualise and evaluate. However, the mental components of workload may be more difficult to measure.

The assessment of your perceived workload for the task is made based on two short tests: rating scales and pairwise comparison of scales to assess the importance of the scales.

Rating Scales

Since workload is something that is explained individually by each performer, there are no effective measures that can be used to estimate workload of different activities. One way to find out about workload is to ask people to describe the feelings they experienced. Because workload may be caused by many different factors, we would like you to evaluate several of them individually rather than lumping them into a single global evaluation of overall workload. This set of six rating scales was developed for you to use in evaluating your experience during the task. Please read the descriptions of the scales carefully. If you have a question about any of the scales in the table, please ask. It is extremely important that they are clear to you. You may keep the descriptions for your reference during the experiment.

After performing the task, you will be given a rating sheet of rating scales. You will evaluate the task by putting an "X" on each of the size scales at the point that matches your experience. Each line has two endpoint descriptors that describe the scale. Please consider your responses carefully in distinguishing among the different task conditions, and consider each scale individually.

Sources of Workload

Following the completion of this experiment, the rating scales are used to assess your experiences in the task. Scales of this sort are useful, but their usefulness suffers since people interpret them in different ways. For example, some people feel that the mental or temporal demands are the essential aspects of workload regardless of the effort they expend on a given task or the level of performance they achieved. Others feel that if they performed well the workload most have been low and if they performed badly it must have been high. Yet others feel that effort or feelings of frustration are the most important factors in workload, and so on. Previous studies have found every possible combination, and the relative impact these factors can be affected by the task being undertaken. For example, some tasks might be difficult because they must be completed very quickly, others may seem easy or hard because of the intensity of mental or physical effort required. Yet others feel difficult because they cannot be performed well, no matter how much effort is expended.

The second stage is a technique that has been developed by NASA to assess the relative

importance of six factors in determining how much workload you experienced. The procedure is simple: you will be presented with a series of pairs of rating scales, and asked to choose which of the items was more important to your experience of workload in the task that you just performed.

After you have finished the entire series we will be able to use the pattern of your choices to create a weighted combination of the ratings from that task into a summary workload score. Please consider your choices and make them consistent with how you used the rating scales. There is no correct pattern, we are interested in your opinion.

F.10 Post-Study Questionnaire

1. Please give any comments you have on the design you created.

2. Please indicate *how* you created the design.

3. Please give comments about the tool you used to create the design.

If you want to have the study as a whole explained to you, please do so now. However we ask that you refrain from discussing this with potential future participants.

G Design Study Analysis

This appendix contains the figures that are supplementary to the analysis of the results of the design space exploration study, in Section 6.3.

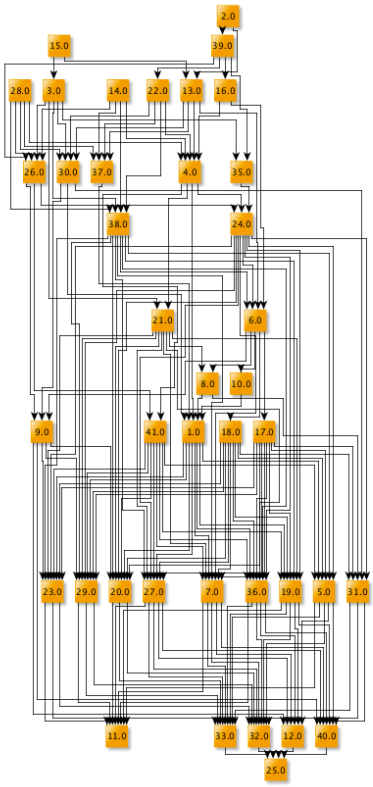
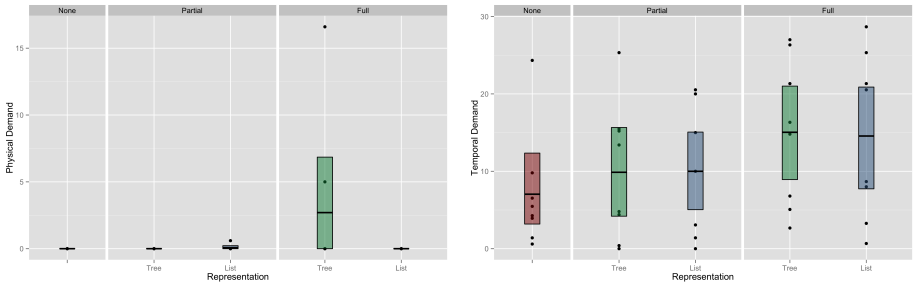


Figure G-2: Hierarchically-ordered specificity relations



(a) Physical demand (b) Temporal demand

Figure G-3: Mental and temporal demand sub-scores across levels of DS support ($R = 1000, 95\%c.i.$)

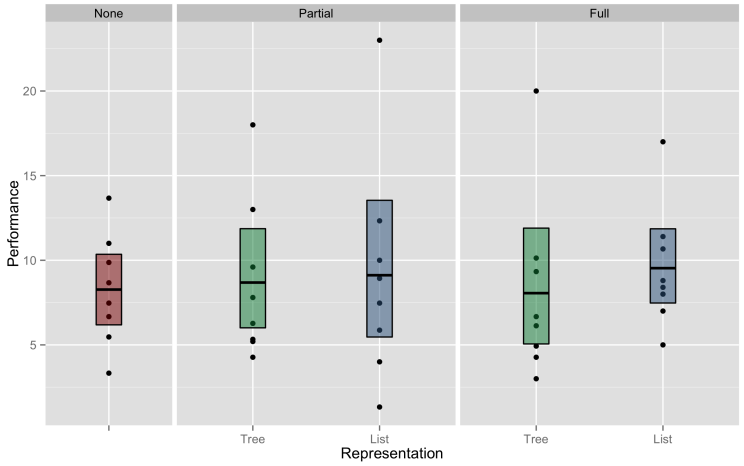


Figure G-4: Performance scores across conditions ($R = 1000, 95\%c.i.$)

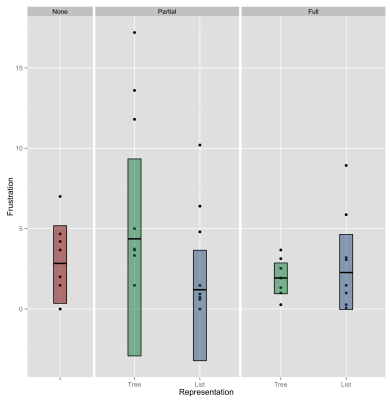


Figure G-5: Frustration across levels of DS support ($R = 1000, 95\%c.i.$)